Cognitive Resource Optimization for the Decomposed Cloud Gaming Platform

Wei Cai, *Student Member, IEEE*, Henry C. B. Chan, *Member, IEEE*, Xiaofei Wang, *Member, IEEE*, and Victor C. M. Leung, *Fellow, IEEE*

***Abstract*— Contrary to conventional gaming-on-demand services that stream gaming video from cloud to players' terminals, a decomposed cloud gaming platform supports flexible migrations of gaming components between the cloud server and the players' terminals. In this paper, we present the design and implementation of the proposed decomposed gaming system. The cognitive resource optimization of the system under distinct targets, including the minimization of cloud, network, and terminal resources and response delay, subject to quality of service (QoS) assurance, is formulated as a graph partitioning problem that is solved by exhaustive searches. Simulations and experimental results demonstrate the feasibility of cognitive resource management in a cloud gaming system to efficiently adapt to variations in the service environments, such as increasing the number of supported devices and reducing the network bandwidth consumption of user terminals, while satisfying different QoS requirements for gaming sessions. We also suggest two heuristic algorithms based on local greedy and genetic algorithm approaches, which can potentially provide scalable but suboptimal solutions in large-scale implementations.**

***Index Terms*— Cloud, cognitive, decomposed, resource management, video game.**

I. Introduction

THE video game industry, driven by the huge sales of mobile games, video gaming consoles, and computer gaming software, has a major impact on the economy. The rapid development of cloud computing technology and its novel concept of providing Everything as a Service [1] is revolutionizing this industry and moving it into a new era. Recently, researchers, system designers, and application developers have become more and more interested in the emerging research topic of cloud gaming [2], which transforms the traditional gaming software into Gaming as a Service (GaaS) [3]. In contrary to conventional video games, the cloud gaming model exhibits several unique advantages.

1) *Scalability:* It overcomes the resource constraints of gaming terminals, including processing capacity, data storage, and battery energy in mobile devices.
2) *Cost Effectiveness:* It reduces the production cost with a unified development approach.
3) *Ubiquitous and Multiple-Platform Support:* It provides cross platform and seamless gaming experience.
4) *Effective Antipiracy Solution:* It provides a potential solution to the troublesome piracy problems and transforms the game developing companies to game service providers.
5) *Click and Play:* It supports a play-as-you-go mode, in which a player can start a game session without downloading or installing the complete game software.
6) *Energy Efficiency:* It has strong potentials in saving batteries of the mobile terminals and bringing longer gaming times for the players by offloading game programs' high computational complexity to the cloud. Therefore, the development of cloud gaming solutions is currently of significant interest not only to academia but also to gaming, cloud computing, and networking industries.

Some game companies such as OnLive,[1] Gaikai,[2] and G-Cluster[3] have started to provide commercialized cloud gaming services to the public. Following a remote rendering GaaS (RR-GaaS) or gaming-on-demand model, game service providers host their video games in cloud servers and stream the players' gaming video frames to their terminals over the Internet. On the other hand, gaming interactions triggered by game players are transmitted to the cloud server over the same networks [4]. With this approach, the cloud gaming service enables the players to run sophisticated games despite the capacity limitation of their mobile devices, at the expense of higher communications and network consumptions. It is obvious that the workload of gaming video rendering in the cloud is extremely heavy, and the video frame transmissions via the Internet also consume huge amounts of network resources, which can lead to high delays in gaming responses [5], [6]. Even though researchers have put plenty of efforts to optimize

Manuscript received September 22, 2014; revised February 4, 2015 and April 12, 2015; accepted June 11, 2015. Date of publication June 26, 2015; date of current version December 3, 2015. This work was supported in part by the University of British Columbia Four Year Doctoral Fellowship, in part by the Natural Sciences and Engineering Research Council of Canada under Grant STPGP 447524, and in part by the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. This paper was recommended by Associate Editor K.-T. Chen.

W. Cai is with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada, and also with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: weicai@ece.ubc.ca).

H. C. B. Chan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: cshchan@comp.polyu.edu.hk).

X. Wang 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: xfwang@ece.ubc.ca; vleung@ece.ubc.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2015.2450171

[1]http://www.onlive.com
[2]http://www.gaikai.com
[3]http://www.g-cluster.com

1051-8215 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

the RR-GaaS systems [7]–[10], the constraints imposed by existing means of Internet access are hindering the wide adoption of gaming on demand.

With improvements in hardware performance, most gaming terminals, including mobile devices, are capable of performing complicated graphical rendering for game scenes. Under this circumstance, a mechanism for on-device rendering instruction execution, also known as local rendering GaaS (LR-GaaS), where only instructions needed to render the images are sent to clients instead of real-time video frames, could be adopted. Such a mechanism is used in [11], which proposes a virtual display architecture for thin-client computing. In this architecture, applications are hosted in a remote server with a virtual device driver that simulates user inputs received from client devices and a virtual display driver that intercepts and transmits drawing commands of screen updates over the network to client devices. However, for a typical LR-GaaS model, e.g., browser games [12], the workload in the cloud is still very heavy, especially when the number of users increases to a certain level. Moreover, the LR-GaaS model is not flexible; in fact, some procedures other than graphical rendering, e.g., frequent and simple calculations, can be executed locally to reduce the latency of responses and also reduce the cloud workload.

Decomposition provides a potential solution for cloud-based online gaming. An architecture has recently been proposed in [13] to decompose the rendering module of a game into two submodules: one submodule executing in the cloud to render the scenes and create the rendering instructions for game scene updates, while another submodule interprets the rendering instructions and transmits them to the mobile devices for local execution. By decoupling the creation of rendering instructions from its execution and transmitting only small-sized rendering instructions over the Internet, the communication burden caused by video transmissions is eased and hence meeting the challenges caused by the limitations of the mobile networks. As a further step, the novel idea to decompose the game program into inter-dependent components that can be distributed to either the cloud or local terminal for execution was first introduced in [14], which achieves flexible resource allocation. Accordingly, the adaptive, interactive, contextual, iterative, and stateful paradigm of cognitive computing [15] has motivated us to build a context-aware platform that adapts the cloud-terminal workload to the runtime environment to optimize the use of cloud, network, and terminal resources while meeting the quality of service (QoS) objectives of the gaming sessions.

We highlight the important contributions of this paper in the following.

1) *System Design and Implementation:* We present a novel design for decomposed cloud gaming platform, provide the framework description, and address the system implementation issues of measuring component execution performance and communication status. We also develop prototype games to conduct empirical evaluations.
2) *System Modeling and Formulation:* To the best of our knowledge, this is the very first work in modeling cognitive resource management for cloud gaming, which builds a foundation for further research. We provide the formulation of a graph partitioning problem and investigate the QoS requirements in cloud gaming scenarios to study the different optimization targets for the proposed system.

The remaining sections of this paper are organized as follows. We review related work in Section II and present the proposed system design and implementation in Section III. Then, in Section IV, we model and formulate the decomposed cloud games as a graph partitioning problem and perform theoretical analysis on optimal solutions. The QoS requirements and the optimization targets for cognitive cloud resource management are described in Section V. We further suggest two heuristic approaches for scalable implementations, based on local greedy and genetic algorithm (GA) approaches, in Section VI. The results of simulations and experiments conducted over a test bed are presented in Sections VII and VIII, respectively. Section IX concludes this paper.

II. Related Work

A. Dynamic Partitioning

The resource allocation optimization problem considered in this paper belongs to a group of dynamic partitioning problems for multiple devices. Studies on the dynamic partitioning between cloud and users' mobile terminals have been conducted for general-purpose applications. Giurgiu *et al.* [16] first introduced a K-step algorithm to compute partitioning on the fly, when a phone connects to the server and specifies its resources and requirements. Furthermore, Chun and Maniatis [17] formulated the dynamic partitioning problem and discussed the supporting platform that facilitates it. A dynamic partitioning system named CloneCloud was designed in [18]. As a flexible application partitioner and execution runtime, CloneCloud 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. Similar to MAUI [19], CloneCloud partitions applications using a framework that combines static program analysis with dynamic program profiling and optimizes execution time or energy consumption using an optimization solver. For offloaded execution, MAUI performs method shipping with relevant heap objects, but CloneCloud migrates specific threads with relevant execution state on demand and can merge migrated state back to the original process. However, a basic requirement of these two works is that identical application copies must be placed in both cloud and terminal sides *a priori*, which conflicts with our design principles of click-and-play for the cloud-based games. Moreover, the mandatory static analysis of both MAUI and CloneCloud requires all programs to be hosted *a priori* in the platform for the static analysis. This procedure extracts the relationships of components, which is necessary information for dynamic profiling and also potentially increases the efficiency of dynamic adaption. However, the static analysis needs additional code instruments, which further complicates the program development. In contrast, on-the-fly adaptation simplifies the program development, while its real-time estimation and optimization features may introduce system overheads, such as increased resource consumption and latency. Our proposed platform supports both static and on-the-fly partitioning, since a dynamic measurement method for the components' performances is introduced. Moreover, we also enable cross-platform gaming experience by adopting the JavaScript language.

B. Cognitive Cloud Gaming Platform

Cai *et al.* [20] were the first to propose a component-based cognitive gaming platform that enables distributed component execution for the purpose of supporting both click-and-play and cognitive resource allocation. The proposed game design also allows the players to continue their gaming sessions in particular gaming scenes without network connection. The cognitive gaming platform is able to dispatch selected gaming components from cloud to players' terminals and later facilitate dynamic partitioning to adapt the QoS to the real-time systemic environment. As a development-friendly environment, the platform also provides a set of application programming interfaces so that the game developers need not understand the lower layer resource management details and focus on the design of game programs. However, system modeling, optimization problem formulation, and proof-of-concept implementations were missing in this previous work.

C. Quality of Experience in Cloud Gaming

Maintaining an acceptable quality of experience (QoE) for the game players is an imperative design concern for cloud gaming systems. To provide cloud gaming service, the relationships between cloud gaming QoE and QoS are different for different implementation architectures. For RR-GaaS, many subjective user studies have been conducted to demonstrate the relationships between cloud gaming QoE and QoS, including game genres, video encoding factors, central processing unit (CPU) load, memory usage, link bandwidth utilization [21], response latency and the game's real-time strictness [22], category of gaming scenes [23], and network characteristics (bit rates, packet sizes, and inter-packet times) [24], [25]. An empirical network traffic analysis of OnLive and Gaikai has been presented in [26]. Nevertheless, with the proposed cognitive cloud gaming platform, the QoE to QoS mapping needs to be redefined due to the adaptive nature of the platform. There is a relatively little research done in this respect. Therefore, in this paper, we focus on hard QoS requirements.

III. System Overview

A. System Design

Fig. 1 shows the design of the cognitive platform that facilitates the dynamic partitioning. *Execution Monitor*s implemented on both cloud and mobile sides monitor the execution information of each component in the cloud, access network, and mobile terminal. The surveillance data, including memory usage, CPU percentage, and execution time, are reported to the *cognitive decision engine*, where cognitive

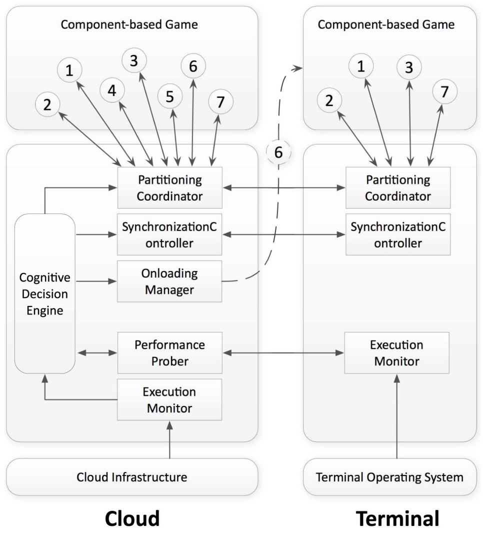


Fig. 1. Cognitive platform for mobile cloud gaming.

resource management strategies are made. The *cognitive decision engine* also requests information from the *performance prober*, which periodically reports its results in probing the cloud-network-terminal environmental parameters. Games designed for the cognitive platform consist of a number of inter-dependent game components. These components are able to migrate from the cloud to the mobile terminal via network under the instruction of the *onloading manager*. Serving as a message gateway between components, the *partitioning coordinator* intelligently selects destination components, locally or remotely, to achieve dynamic resource allocation. The *synchronization controller* is designed to guarantee the synchronization of data in identical components distributed in the cloud and mobile terminals. Note that the *onloading manager*, *partitioning coordinator*, and *synchronization controller* work with the *performance evaluator* and *local analyzer* for the purpose of maintaining an acceptable QoS for players.

B. Dynamic System Monitoring

One of the most critical problems for the decomposed cloud gaming platform is to design a practicable mechanism to measure the execution status, e.g., component execution costs, execution probability, and communication costs. In our design, we introduce an *execution monitor* and a latency-based estimation solution to derive these parameters. The *execution monitors* on both the cloud and mobile sides monitor resource usage of each components, whether in cloud or terminal, and save the execution information in a statistics database. This execution information includes the property of each component in each invocation, including memory consumption, CPU usage percentage, execution time, output data size, and execution environment. 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 are identified from this database. The *performance prober* is a software mobile agent [27] that travels between cloud and terminals to estimate the cloud-network-terminal environmental parameters. 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 terminals. Its execution information is measured and report to the performance prober in the cloud. Note that this process involves two network transmissions, in which the system is able to calculate the network QoS parameters, including round-trip time (RTT) 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 executional information for all components with computation. Denoting the cloud computational efficiency as E_c and 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 measure from the system. For instance, the cloud execution time for a particular component can be estimated as $E_t \times R_E$. To minimize the overhead of probing, the performance prober is designed to work collaboratively with the execution monitor. As mentioned before, the cognitive system follows a stochastic approach to calculate the execution probability of each component. Hence, both cloud's and terminals' databases need to be queried. Meanwhile, the statistics in the terminal are 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. Of course, our design imposes some overheads, including recording each component's invocation in real time, calculating components' invocation frequency from the records and periodic end-to-end network transmission of the performance prober mobile agent. However, these computational overheads are negligible in current hardware, while the network transmission overheads are ineradicable in all existing cognitive systems. In summary, these costs are minimized by the reuse of status monitoring for networks and terminals. In our design, the system dispatches performance prober to the terminals in an interval of T_I and save the probing results in a database. In fact, with real-time data analysis, the interval T_I can also be a variable subject to the variety of the terminal, e.g., the network quality.

IV. System Modeling

In this section, we model the resource management problem from the perspectives of game components, QoS constraints, and optimization targets.

A. Game Components

In the cognitive cloud gaming platform, games consist of inter-dependent components that work collaboratively to provide gaming services for players. As shown in Fig. 2, we model the dependent game components by a directed graph $G = \{V, E\}$, where each vertex in V represents a game component v_i and each edge $e_{i,j}$ in E indicates a dependency

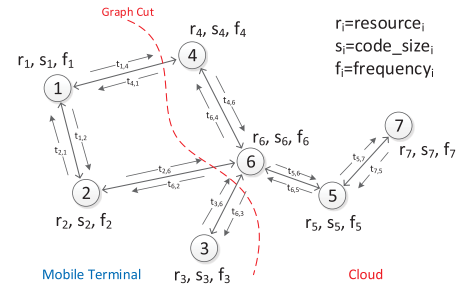


Fig. 2. Components partitioning for the proposed cognitive platform.

TABLE I
Modeling Notation

Symbol	Definition
r_i	the resource consumption of v_i
s_i	the size of the compiled code of v_i
f_i	the execution frequency of v_i
$o_{i,j}$	the amount of output data from v_i to v_j
$ft_{i,j}$	the frequency that v_i sends data output to v_j

between v_i and v_j. Each component v_i is characterized by the parameters listed in Table I.

Once the mobile terminal fetches the game components from the cloud, the *partitioning coordinator* works with the *cognitive decision engine* to solve the dynamic partitioning problem to provide a QoS-oriented resource optimization. In this framework, all input and output data from the components are sent to the *partitioning coordinator*, which provides a routing service to invoke messages by intelligently selecting the destination components when an application cycle is determined.

As shown in Fig. 2, the partitioning problem intrinsically seeks to find a cut in the component graph such that some components of the game run on the client side and the remaining ones run on the cloud side. The optimal cut maximizes or minimizes an objective function O, which expresses the general goal of a partition such as minimizing the end-to-end interaction time between the mobile terminal and the cloud, minimizing the resource consumption in the cloud or minimizing the data transmissions for the game terminals. Therefore, the resource management problem is to seek an optimal set of all connecting terminals' partitioning solutions in discretized search space, which meets the system's optimization target.

B. Formulation

To formulate the costs associated with partitioning of the game components, whose properties are listed in Table I, we further define specific parameters, including resource consumptions, code migration cost, and output transmission cost, as shown in Table II. Note that the term cost in this paper is used as a measure of resource utilization or consumptions in terms of how they impact system performance. For our practical implementation, the cost is given by the actual latency.

Note that t_i and c_i denote the component execution costs in the terminals and the cloud, which give an overall evaluation

TABLE II
FORMULATION NOTATION

Symbol	Definition
σ	the set of components allocated in cloud
τ	the set of components allocated in terminal
t_i	execution cost of v_i in terminal
c_i	execution cost of v_i in cloud
m_i	migration cost of v_i from cloud to terminal
p_i	execution probability of v_i
$\alpha_{i,j}$	communication cost from v_i to v_j, $i, j \in \sigma$
$\beta_{i,j}$	communication cost from v_i to v_j, $i, j \in \tau$
$\lambda_{i,j}$	communication cost from v_i to v_j, $i \in \sigma$, $j \in \tau$
$\theta_{i,j}$	communication cost from v_i to v_j, $i \in \tau$, $j \in \sigma$
$q_{i,j}$	probability that v_i sends data output to v_j

of resource consumptions r_i in CPU, memory, and energy, m_i denote the overall transmission cost of migrating component i from cloud to terminal, which depends on code size s_i, network quality, and expected gaming session time, and $\alpha_{i,j}$, $\beta_{i,j}$, $\lambda_{i,j}$, and $\theta_{i,j}$ denote message communication costs between components, subject to the amount of output data $o_{i,j}$ and network QoS parameters. In addition, p_i is subject to f_i, and $q_{i,j}$ is subject to $f_{i,j}$. Accordingly, we formulate the optimal partitioning problem to minimize the overall cost C_o, which is a sum of execution costs for all components C_e, communication costs between all components C_c, and code migration costs C_m as

$$C_o = C_e + C_c + C_m. \tag{1}$$

According to our definitions in Table II, we derive

$$C_e = \sum_{v_i \in \tau} t_i p_i + \sum_{v_i \in \sigma} c_i p_i \tag{2}$$

$$C_c = \sum_{\substack{v_i \in \sigma \\ v_j \in \sigma}} \alpha_{i,j} q_{i,j} + \sum_{\substack{v_i \in \tau \\ v_j \in \tau}} \beta_{i,j} q_{i,j} + \sum_{\substack{v_i \in \sigma \\ v_j \in \tau}} \lambda_{i,j} q_{i,j} + \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} \theta_{i,j} q_{i,j} \tag{3}$$

$$C_m = \sum_{v_i \in \tau} m_i. \tag{4}$$

C. Optimal Partitioning Solution

The minimum overall cost C_o is subject to the various costs parameters denoted in Table II. In this section, we derive the minimum cost for a special case, where the parameters for all v_i and e_i in Table II are all identical: we denote their constant values by $t, c, m, p, \alpha, \beta, \lambda, \theta$, and q, respectively. Consequently, the directed graph for component partitioning is transformed to a connected and undirected graph with identical weight $(\lambda + \beta) \cdot q$ for the edges. We derive the minimum cost for different graph topologies with N components, including the minimum spanning tree (MST), complete graph, and general graph. As a practical constraint, at least one component is executed in each player's terminal, acting as the player's command receiver and transmitter.

1) Partitioning for Minimum Spanning Tree: An MST or minimum weight spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree.

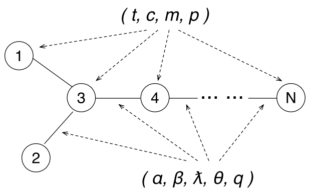


Fig. 3. Minimum spanning tree with N components.

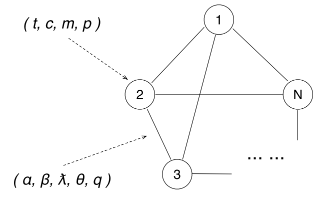


Fig. 4. Complete graph with N components.

In our context, it refers to a connected graph topology that contains $N - 1$ edges, as shown by the example in Fig. 3.

Assume $k \in [1, n-1]$ cuts in the MST $G_M(N)$ to split the N components so that $x \in [1, N-1]$ components are placed in the terminal and $N - x$ components are placed in the cloud. This leads to the following cost functions:

$$C_e = x \cdot t \cdot p + (N - x) \cdot c \cdot p \tag{5}$$

$$C_m = m \cdot x \tag{6}$$

$$C_c = [(x-1) \cdot \beta + (N - x - 1) \cdot \alpha + k \cdot (\lambda + \theta)] \cdot q. \tag{7}$$

Since $(\lambda + \beta) \cdot q \geq 0$, to minimize the C_o, the value of k should be set to 1, the minimum value. The derivative of the function C_o at x is shown in

$$C_o' = [(t - c) \cdot p + m + (\beta - \alpha) \cdot q] \times 2. \tag{8}$$

Therefore, we see that if $C_o' \geq 0$, the value of x should be the minimum, i.e., 1, to minimize C_o. On the other hand, if $C_o' < 0$, the value of x should be the maximum, i.e., $N - 1$. In general cases, the terminals' computational efficiency is always lower than the computational efficiency of the cloud. Hence, we define $t > c$ and $\beta > \alpha$, which makes $C_o' \geq 0$. Accordingly, the optimal partitioning for MST is only to migrate the necessary components to the terminal, while executing all of the others in the cloud. However, if the cloud server is suffering from an extremely heavy workload, the computational power in terminals may exceed the cloud, then the system should migrate all components to the terminal when $C_o' < 0$, where the parameters satisfies

$$m < (c - t) \cdot p + (\alpha - \beta) \cdot q. \tag{9}$$

2) Partitioning for Complete Graph: A complete graph is a simple undirected graph in which every pair of distinct vertices are connected by a unique edge, as shown in Fig. 4.

In fact, a complete graph $G_C(N)$ with N components contains $N(N-1)/2$ edges. Assume there are $x \in [1, N-1]$ components executed in the terminal and $N - x$ components executed in the cloud, we derive C_e and C_m as in (5) and (6), and C_c is formulated as

$$C_c = [\beta \cdot x(x-1)/2 + \alpha \cdot (N-x)(N-x-1)/2 + (\lambda+\theta)\cdot(N-x)x]\cdot q. \quad (10)$$

Therefore, we derive $C_o = C_e + C_m + C_c$. In general cases, $\alpha, \beta \ll \lambda, \theta$, which leads to $\alpha + \beta - 2\lambda - 2\theta < 0$, which indicates that C_o has a downward slope. Consequently, the minimum value of C_o is when either $x = 1$ or $x = N-1$. Given $C_o = f(x)$, here we derive $\Delta = f(N-1) - f(1)$ as

$$\Delta = (N-2)[q\cdot(\beta-\alpha)(N-1)/2 + p\cdot(t-c) + m]. \quad (11)$$

Given $N > 2$, we see that if $\Delta \geq 0$, to minimize C_o, the value of x should be 1. If $\Delta < 0$, the value of x should be $N-1$. In general cases, the terminals are less computational efficient than the cloud; thus, $t > c$ and $\beta > \alpha$, which makes $\Delta \geq 0$. Accordingly, the optimal partitioning is that only the necessary single component is migrated to the terminal, while all of the others are executed in the cloud. However, if the cloud server is suffering from an extremely heavy workload such that the computational power in terminals exceeds the cloud, the system will host all components at the terminals when $\Delta < 0$, where the parameters satisfy

$$m < (c-t)\cdot p + (\alpha-\beta)\cdot(N-1)\cdot q/2. \quad (12)$$

3) Partitioning for General Graph: A general graph refers to a connected graph topology in which vertices are connected by an arbitrary set of edges. In this section, we discuss the overall cost for an arbitrary connected general graph $G_R(N)$ with N vertices and $\{E|2(N-1) < E < N(N-1)/2\}$ directed edges.

Theorem 1: Given a specific N and that the connected graph G_A is a subgraph of G_B, and G_A and G_B share the same partitioning solution P, the overall cost of C_o satisfies $C_o(G_A) < C_o(G_B)$.

Proof: Since G_A is a subgraph of G_B with a specific N, and G_A and G_B share the same partitioning solution P, we have $C_e(G_A) = C_e(G_B)$ and $C_m(G_A) = C_m(G_B)$. Since G_A is a subgraph of G_B, we denote a link set $L = L(G_B) - L(G_A)$, where L_{G_A} and L_{G_B} are the link sets of G_A and G_B, respectively, then we derive $\Delta C_c = C_c(G_B) - C_c(G_A)$ as

$$\forall v_i, v_j \in L$$
$$\Delta C_c = \sum_{\substack{v_i\in\sigma\\ v_j\in\sigma}} \alpha_{i,j}q_{i,j} + \sum_{\substack{v_i\in\tau\\ v_j\in\tau}} \beta_{i,j}q_{i,j} + \sum_{\substack{v_i\in\sigma\\ v_j\in\tau}} \lambda_{i,j}q_{i,j} + \sum_{\substack{v_i\in\tau\\ v_j\in\sigma}} \theta_{i,j}q_{i,j}.$$

Therefore, when $\Delta C_c > 0$, $C_c(G_B) > C_c(G_A)$ is true. Since $C_o = C_e + C_c + C_m$, we obtain $C_o(G_A) < C_o(G_B)$. The theorem is proved. ■

Theorem 2: Given a specific N and that the connected graph G_A is a subgraph of G_B, the minimal overall cost of C_o^m satisfies $C_o^m(G_A) < C_o^m(G_B)$.

Proof: Denote G_A by a subgraph of G_B, where $C_o^m(G_A)$ is the minimal overall cost of G_A with the optimal partitioning of $P(A)$ and $C_o^m(G_B)$ is the minimal overall cost of G_B with optimal partitioning solution $P(B)$. Suppose $C_o^m(G_A) \geq C_o^m(G_B)$ is true. Trim G_B to $G_{B'}$, where $G_{B'} = G_A$. According to Theorem 1, we have $C_o^m(G_B) > C_o(B')$, where $C_o(B')$ is the overall cost of $G_{B'}$ with the partitioning solution $P(B)$. Therefore, $C_o^m(G_A) > C_o^k(G_A)$, where $C_o^k(G_A)$ is the overall cost of G_A with the partitioning solution $P(B)$. It contradicts the assumption that $C_o^m(G_A)$ is the minimal overall cost of G_A. The theorem is proved. ■

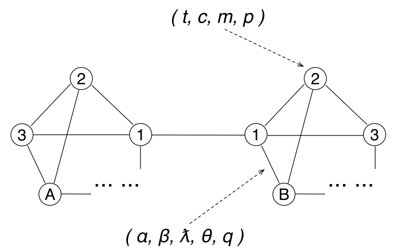


Fig. 5. General graph with N components.

Since $G_M(N)$ is a subgraph of $G_R(N)$ and $G_R(N)$ is a subgraph of $G_C(N)$, according to Theorem 2, we derive

$$C_o(G_M(N)) < C_o(G_R(N)) < C_o(G_C(N)). \quad (13)$$

Take a general graph with N vertices as an example. Assume there is a complete graph with A vertices and another complete graph with $B = N - A$ vertices. Also assume there is one edge between vertex A and B, as shown in Fig. 5. It is mandatory that component A is hosted in the cloud and component B is executed in a terminal.

Assume there are $x \in [1, N-1]$ components executed in the terminal and $N - x$ components executed in the cloud, with C_e and C_m derived in (5) and (6), the minimized $\overline{C_c}$ is formulated as

For $x_l < B$

$$\overline{C_c(x_l)} = \frac{A(A-1)\alpha q}{2} + \frac{(B-x)(B-x-1)\alpha q}{2} + \alpha q + (\lambda+\theta)x(B-x)q + \frac{x(x-1)\beta q}{2}. \quad (14)$$

For $x_e = B$

$$\overline{C_c(x_e)} = \frac{A(A-1)\alpha q}{2} + (\lambda+\theta)q + \frac{B(B-1)\beta q}{2}. \quad (15)$$

For $x_g > B$

$$\overline{C_c(x_g)} = \frac{x(x-1)\alpha q}{2} + (\lambda+\theta)(N-x)(x-B)q + \frac{(x-B)(x-B-1)}{2}\beta q + \frac{B(B-1)}{2}\beta q + \beta q. \quad (16)$$

Given $C_o = f(x)$, we have $\Delta_l = f(x_e) - f(x_l)$. In general, $c \approx t$ and $\alpha, \beta \ll \lambda, \theta$. To this end, we consider the minimum $C_o(x_e) < C_o(x_l)$ when the values of the parameters satisfies

$$m(x_e - x_l) < [x_l(B - x_l) - 1](\lambda+\theta)q. \quad (17)$$

TABLE III
QoS NOTATIONS

Symbol	Definition
$\{d \mid d \in D\}$	the set of supported D terminal devices
$n_{i,j}(d)$	network consumption between v_i and v_j for terminal d
$l_p(d)$	expected process latency for terminal d
$l_c(d)$	expected communication latency for terminal d
$L_M(d)$	maximal tolerable latency for terminal d
$T_{RTT}(d)$	Round Trip Time for terminal d
F_C	cost-time factor for cloud processing
F_T	cost-time factor for terminal processing
F_N	cost-time factor for network communications
$P_T(d)$	available processing resources in terminal d
$N_T(d)$	available network resources in terminal d
P_C	available processing resources in the cloud
N_C	available network resources in the cloud

Similarly, we derive $\Delta_g = f(x_e) - f(x_g)$ as

$$\begin{aligned}\Delta_g &= [(t-c)p+m](x_e - x_g)\\ &\quad + [1-(N-x_g)(x_g-B)](\lambda+\theta)q\\ &\quad + \left[\frac{A(A-1)}{2} - \frac{x_g(x_g-1)}{2}\right]\alpha q\\ &\quad + \left[\frac{(x_g-B)(x_g-B-1)}{2} - 1\right]\beta q. \end{aligned} \tag{18}$$

Since $c \approx t$ and $\alpha, \beta \ll \lambda, \theta$, we can easily derive that $\Delta_g < 0$, and thus minimum $C_o(x_e) < C_o(x_g)$. In conclusion, C_o reaches the minimum value when $x = B$ and $m(x_e - x_l) < [x_l(B - x_l) - 1](\lambda + \theta)q$.

This example illustrates that it is a critical challenge to derive the solution that minimizes the overall cost for a general graph with a variety of edges. The computational complexity of seeking an optimal cut for a general graph is 2^N.

V. CLOUD RESOURCE MANAGEMENT

As an assumption in previous sections, the execution and communication costs are considered as identical constants. Nevertheless, real-world scenarios generally involve various values of execution and communication costs, which further complicate the optimization problem on graph partitioning. In addition, we also need to extend the problem by considering the capacity constraints of cloud and players' terminals. In this section, we formulate the QoS controls in cloud resource management and introduce a set of optimization targets.

A. QoS Constraints

As a gaming service provision system, satisfying all players' QoS is a fundamental requirement. To this end, we formulate the QoS constraints with additional notations in Table III.

1) Terminal Cost Constraint: The terminal device needs sufficient resources to host the set of downloaded gaming components. To simplify our model, we take execution and intra-terminal communication as the consumers of processing resources. The terminal processing resource consumption for terminal d is formulated as

$$\mu(d) = \sum_{v_i \in \tau} t_i(d) p_i(d) + \sum_{\substack{v_i \in \tau \\ v_j \in \tau}} \beta_{i,j}(d) q_{i,j}(d). \tag{19}$$

Then the constraint on terminal processing resource is formulated as

$$\forall d \in D, \quad \mu(d) \le P_T(d). \tag{20}$$

2) Terminal Network Constraint: The remote invokes of components introduce network bandwidth consumption to the gaming procedure. To guarantee the QoS for players, the system needs to control all terminals' gaming throughput. We formulate the network resource consumption between component $v_i \in \tau$ and $v_j \in \sigma$ for terminal d as

$$n_{i,j}(d) = \lambda_{j,i}(d) \cdot q_{j,i}(d) + \theta_{i,j}(d) \cdot q_{i,j} \tag{21}$$

and the constraint on terminal network resource is formulated as

$$\forall d \in D, \quad \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d) \le N_T(d). \tag{22}$$

3) Cloud Resources Constraint: As a service provider, the cloud consumes its processing resources to host a set of components for M terminals. Therefore, to guarantee the continuous resource provisioning is a critical issue in QoS assurance. The cloud processing resource consumption for terminal d is formulated as

$$\nu(d) = \sum_{v_i \in \sigma} v_i(d) p_i(d) + \sum_{\substack{v_i \in \sigma \\ v_j \in \sigma}} \alpha_{i,j}(d) q_{i,j}(d). \tag{23}$$

With the same notations, we formulate the constraint on cloud resources as

$$\sum_{d \in D} \nu(d) \le P_C. \tag{24}$$

4) Cloud Network Constraint: During the gaming session, the cloud handles network connections from the terminals. We also need to formulate the constraint on the overall network throughput as

$$\sum_{d \in D} \left(\sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d) \right) \le N_C. \tag{25}$$

5) Response Delay Constraint: Response delay represents the time interval between players' input and the system response. In cloud gaming system, the latency is caused by *processing latency* and *networking latency*. And hence, in this paper, the expected *processing latency* $l_p(d)$ is formulated as the sum of processing time that is proportional to the sum of processing resource consumptions in both terminal and cloud with the efficient factors F_T and F_C, respectively. Note that to predict and to model the burst of component and communication costs in real-game sessions will be our next step of work

$$l_p(d) = f(\mu(d), \nu(d), F_T, F_C). \tag{26}$$

The expected *networking latency* $l_n(d)$ is formulated as the sum of communication delays for all remote invocations between components, which is proportional to their network resource consumption with the efficient factor F_N

$$l_n(d) = \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} g(n_{i,j}(d), T_{\mathrm{RTT}}(d), F_N) \tag{27}$$

where g is the function to calculate the latency from the specific communication package and RTT. Accordingly, we conclude the constraint on response relay as

$$\forall d \in D, \quad l_n(d) + l_p(d) \leq L_M(d). \tag{28}$$

B. Optimization Targets

As a cognitive system, the cloud gaming service is adaptable by all kinds of terminals, accessible through different networks and can be hosted by clouds provided by different vendors. In this paper, we design a set of optimization targets to demonstrate the flexibility and efficiency of the proposed system.

1) Cloud Resource Cost Minimization: Cloud is not a free resource and its capacity is still restricted by existing visualization techniques. Therefore, minimizing the cloud resource utilization is crucial from the economic perspective. To this end, *cloud resource cost minimization* allocates more components to selected terminals to reduce its own resource consumption. Note that the selecting procedure shall be under the supervision of the QoS constraints, and thus, to guarantee QoS satisfactory for all players as

$$\min \sum_{d \in D} \nu(d)$$
$$\text{s.t. } (20)(22)(24)(25)(28).$$

2) Network Cost Minimization: The cloud gaming largely relies on the network quality. Players who access gaming services via paid mobile network also concern their bill amount. Hence, the system network throughput is an important factor that impacts both players' QoS and users' interests on the cloud gaming service. Consequently, another optimization target *network cost minimization* dynamically determines an optimized partitioning solution to minimize the terminals' average network cost

$$\min \sum_{d \in D} \left(\sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d) \right)$$
$$\text{s.t. } (20)(22)(24)(25)(28).$$

3) Terminal Resource Cost Minimization: Terminal is considered as relatively weak devices, especially when they are mobile devices powered by batteries. Therefore, minimizing the terminal resource utilization is another important optimization factor we need to consider. For this purpose, an optimization target *terminal resource cost minimization* seeks optimal partitioning solution to minimize the terminals' average resource cost and thus provides lower energy consumption rate on players' devices

$$\min \sum_{d \in D} \mu(d)$$
$$\text{s.t. } (20)(22)(24)(25)(28).$$

4) Response Delay Minimization: The response delay to players' commands in gaming systems is defined as the time difference between the time when player initiates a command and the time when player receives the response from the game program. As one of the most key factors that impacts the players' QoS, the response delay shall be strictly controlled under a threshold (e.g., 200 ms is maximal tolerable and 120 ms is hardly noticeable, as stated in [28]) to ensure acceptable gaming procedure. As indicated in [29], the calculation of the response delay is completely different from that in the conventional cloud gaming system. We provide *response delay minimization* mode to minimize the terminals' average response delay with selected partitioning solutions

$$\min \sum_{d \in D} l_n(d) + l_p(d)$$
$$\text{s.t. } (20)(22)(24)(25)(28).$$

VI. Heuristic Solutions

The optimizations presented in the previous section may not scale to more complex systems with large numbers of game players due to the computational complexity of searching for the optimal solutions. Suppose the cloud-based game consists of C components and provides its cloud gaming service for N terminals, the number of potential component allocation solutions is 2^{NC}, which implies that an exhaustive search approach has an extremely high computational complexity. To address this issue, we sketch two possible heuristic solutions here: a local greedy approach and a more sophisticated and efficient GA-based [30] approach, which have the potential to realize the advantages of the cognitive resource optimization proposed in this paper in real-time scalable implementations.

The intuitive motivation of considering a local greedy approach, which partitions the optimization problem into a set of subproblems for all terminals, is its computational simplicity. After solving these subproblems separately, the system concatenates all of the sub-solutions into a complete solution for the global problem. The computational complexity for local greedy approach is significantly reduced to 2^N. Note that since users are not independent of each other in the use of system resources, the concatenated solution might not give the global optimal solution.

In contrast, the GA approach evolves a series of potential solutions, in which each described by a chromosome represents a particular genetic instance of the system, toward a desirable solution. The potential benefits of the GA approach include: 1) controllable computational complexity for quick responses; 2) potential parallel chromosome operations in distributed cloud computing data centers; and 3) possible reuse of high-quality chromosome candidates in different circumstances to speed up the convergence. In the cognitive platform considered in this paper, we can devise a chromosome that consists of $N \times C$ bits, each of which is assigned a value of 1 if the corresponding component is hosted in the cloud or 0 if the component is executed locally in the respective terminal. Thus, the system places a component for execution, e.g., the Yth component of terminal X, by looking up the

TABLE IV
PARAMETERS OF DECOMPOSED GAMES

Parameter	**Value**
Number of Components	10
Component Execution Cost (Cloud)	1∼50
Component Execution Cost (Terminal)	1.05∼50.25
Component Execution Probability	0.1∼0.7
Component Migration Cost	1∼10
Communication Probability	0.25
Data Transmission Frequency	0.01∼0.3
Communication Cost (Cloud to Terminal)	0.1∼200
Communication Cost (Terminal to Cloud)	0.105∼310
Communication Cost (Cloud to Cloud)	0.002∼4
Communication Cost (Terminal to Terminal)	0.003∼6

bit $C * (X - 1) + Y$ in the chromosome. Based on above chromosome encoding, the conventional operators of a GA [30], e.g., cloning, crossover, mutations, and fitness function, and be applied in the system design.

The two proposed heuristic algorithms can be executed at either the run-time (on the fly) or prepublished stage (static approach) of a game. In contrary to on-the-fly execution, the static approach simulates offline all possible combinations of the network and terminal states on a testing platform in the prepublished stage, to find the preferred solutions for the game in any given set of conditions. With this approach, we can create a dictionary that enables the run-time system to pick up the preferred solution according to the states of the system (cloud, network, and terminal) in real time.

VII. SIMULATIONS

A. Simulation Setup

To validate the performance of the cognitive resource management of the proposed gaming platform, we set up the following simulations. Regarding the calculation of the *processing latency* $l_p(d)$ and *networking latency* $l_n(d)$, we assume the functions f and g described in Section V-A follow:

$$f(\mu(d), \nu(d), F_T, F_C) = \mu(d) \cdot F_T + \nu(d) \cdot F_C \quad (29)$$

$$g(n_{i,j}(d), T_{\text{RTT}}(d), F_N) = n_{i,j}(d) \cdot T_{\text{RTT}}(d) \cdot F_N. \quad (30)$$

We design a set of random components and random communications to simulate a decomposed cloud gaming system. The default parameters for the decomposed game is as shown in Table IV. Note that *communication probability* defines the probability for a communication occurs between two components, while *data transmission frequency* denotes the transmission frequency between two components, given there are communications existed. We can also notice from Table IV that the communication cost from terminal to cloud is a bit larger than the one from cloud to terminal, since we assume that the terminal needs to cost more energy to initiate a data transmission, especially when they are powered by batteries.

The default simulation parameters of terminal devices are listed in Table IV, which represent the wide range of terminals, from stationary computers to mobile devices connected to wireless networks.

TABLE V
PARAMETERS OF TERMINAL DEVICES

Parameter	**Value**
Round Trip Time (T_{RTT})	10ms∼25ms
Available Network Resource	200∼500
Available Computing Resource	100∼400
F_C	0.001
F_T	0.002
F_N	0.005

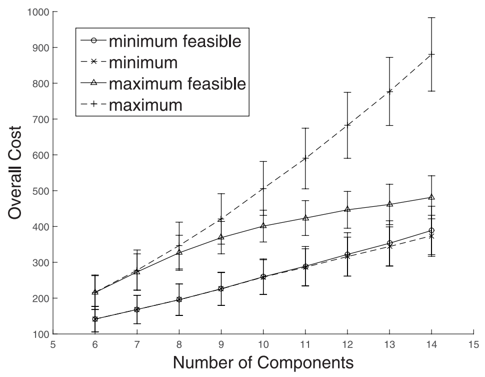


Fig. 6. Effect of the number of components on the overall cost.

All random variables listed in Tables IV and V follow the uniform distribution. To simplify the QoS measurement, we set 120 ms as the maximal tolerable latency value as the indicator of QoS. All simulations are repeated 2000 times with distinct random seeds to yield an average value.

B. Discussion on Game Design

In our experiments, we realize that the game design, including the resource consumption of each component and the data communication between components, will substantially impact the system performance. Hence, the first part of our experimental work is to investigate the features of diverse genres of games to further explore the optimization potential of the cloud-based gaming system. In the simulation model, we first perform the cost minimization to come up with a candidate partitioning scheme and then check whether this candidate solution is feasible or not in terms of whether it meets the QoS constraints.

1) Overall Cost Boundary: In this section, we study the overall cost of a specific game in different partitioning solutions. As shown in Fig. 6, the overall cost increases along with the number of components. However, the increasing speed of the maximum value is faster than the minimum value. Note that not all partitioning schemes are feasible in our proposed system, from the perspective of the QoS restriction. Therefore, we also depict those maximum and minimum feasible values for distinct number of components. As we can see from Fig. 6, the gap between maximum and minimum feasible values is becoming smaller when the number of components increases.

The similar trend is also discovered in Fig. 7, which shows the effect of component communication probability on the

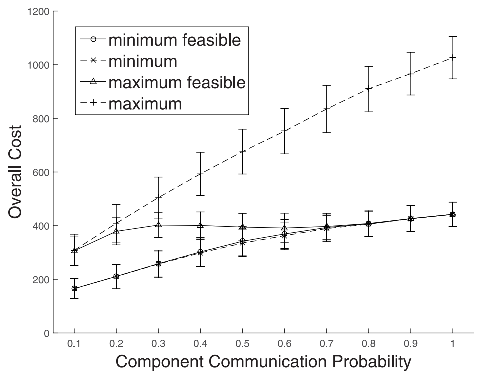


Fig. 7. Effect of component communication probability on the overall cost.

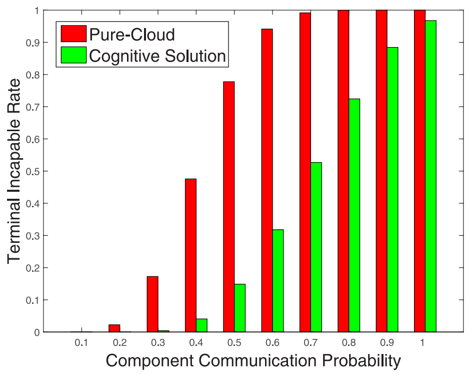


Fig. 8. Comparison on the terminal incapable rate.

overall cost. Apparently, increased communication probability indicates the increases on network costs in optimal partitioning, which excludes more infeasible schemes with higher overall cost from the results. We also plot the error bar for Fig. 6 and Fig. 7 with a confidential interval of 95%.

2) Effect on Terminal Incapable Rate: In our simulation, the games' component structures are randomly created. Hence, some of them might not have feasible partitioning scheme to fulfill the QoS requirements. In this paper, we define *terminal incapable rate* to indicate the ratio that the cloud server and terminals are incapable of supporting a game session. For various of game structures, we compare *PureCloud*, the conventional partitioning solution for cloud gaming, with our proposed cognitive solutions over the *terminal incapable rate*. As shown in Fig. 8, as the communication probability arises, more network communications cause higher overall response delay. For the component communication probabilities of 0.5 and 0.6, the terminal incapable rate for *PureCloud* dramatically increased to 0.78 and 0.95, while our cognitive solution provides a much lower rate at around 0.15 and 0.3. This illustrates the advantage of utilizing cognitive resource allocation for decomposed cloud games: more game genres are supported.

C. System Performance Evaluation

In this section, we evaluate the system performance based on the two proposed algorithms. The default simulation parameters for genetic evolution are listed in Table VI.

TABLE VI
PARAMETERS FOR GA SIMULATION

Parameter	Value
Population	50
Evolution Iterations	100
Expanding Factor	2
Crossover Ratio	0.9
Mutation Ratio	0.3
Cloud Network Resource	12000
Cloud Computing Resources	6000
Number of Game Components	10

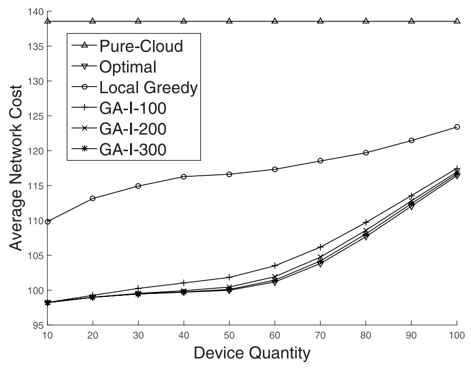


Fig. 9. Performance evaluation on network cost minimization.

1) Network Cost Minimization: We first evaluate the performances of local greedy approach and GA solution regarding the network cost minimization. With extensive random seeds, we derive the average network costs achieved by a local greedy algorithm and a GA with various generation iterations of 100, 200, and 300, respectively. For comparison, we also illustrate the average network cost of the *PureCloud* solution and optimal, the systemic minimal average network cost. Note that to eliminate 2^{CN} search for optimal solutions, we set up an infinite loop for GA iteration. If an additional 2000 iterations yields the same result, we consider that the GA converges to the optimal solution. This value is then adopted as the optimal value. Note that the local greedy and GA running times for different iterations increase along with the quantity of devices. We measure these execution times in our ASUS windows 7 personal computer (PC) with Intel Pentium G630 @ a 2.70-GHz central processor unit (CPU) and a 4.0-GB internal memory (RAM). According to our experimental settings, even though for 100 devices, the local greedy algorithm and GA with 100 and 200 iterations can be completed within 1 s, while the GA of 300 iterations can be done in 2 s. In contrast, the 2000 iterations of GA consume around 28 s with 100 devices, while only 1 s for 10 devices' scenario. In fact, the algorithms' running time depends on the hardware capacity. If we upgrade the computer and apply parallel processing, these execution times can be shortened further.

As shown in Fig. 9, with large-scale simulations, the local greedy algorithm provides around 19.5%–10.5% decrease in average network cost compared with the *PureCloud* solution, while the series of GA solutions demonstrates even higher efficiency in optimizing the average network cost. It can be

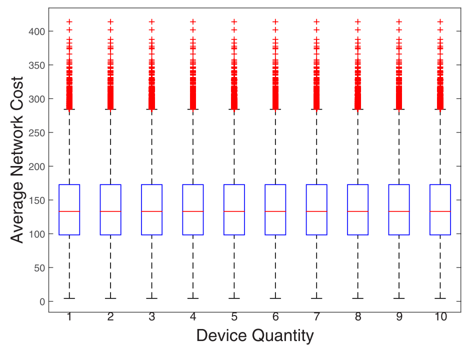


Fig. 10. Variance of the optimal scheme simulation results.

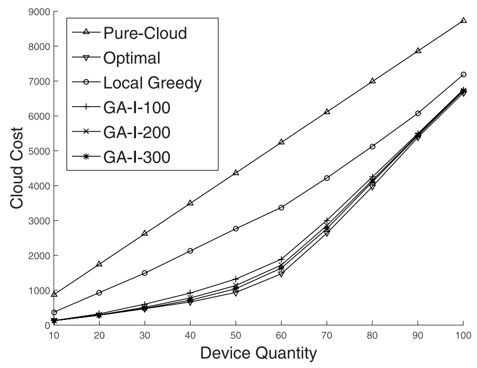


Fig. 11. Performance evaluation on cloud resource cost minimization.

observed that when the device quantity is relatively small, e.g., 10, a small number of generation iterations (e.g., 100) are able to achieve optimal solution with a 27.6% cost decline. On the other hand, when the device quantity increases, more generation iterations for GA are required to approach the optimal results. Note that optimized average network cost grows as the device quantity increases. The reason is that the system needs to sacrifice the performance to fulfill the requirements of supporting more terminals: some of the optimal partitioning with minimum response delay might not be qualified as feasible solutions. In our experimental settings, with 6000 overall cloud computing resources, the optimization on average network cost will be affected when the quantity of terminal devices is larger than 60. To clearly demonstrate the variance, we select the optimal scheme as representative data to show a box plot in Fig. 10, where the bottom and top of the boxes are always the first and third quartiles, respectively, and the band inside the boxes is always the second quartile (the median).

2) *Cloud Resource Cost Minimization:* We also perform simulation on the optimization target of cloud resource cost minimization to compare the efficiencies of *PureCloud* solutions, local greedy algorithm, and a series of GA schemes.

Fig. 11 shows the comparison on the average throughput of the terminal devices. Apparently, for *PureCloud* mode, the cloud cost is proportional to the device quantity. To support 100 terminals, the gaming server need to request around 9000 units computational resource. In contrast, with cloud resource cost minimization, the overall cost of the cloud is significantly reduced: to achieve the optimal value, only

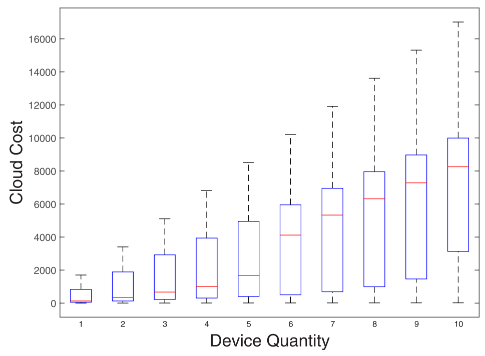


Fig. 12. Variance of the local greedy scheme simulation results.

around 6800 units are required, which is only 75.5% of the conventional *PureCloud* mode. Given the higher computational complexity of iterative GA solution, the local greedy approach also brings the cloud cost down to 3000 units, which yields a 22.2% decline in our experimental settings. Similar to the network cost minimization, the cloud resource cost minimization is also constrained by the QoS requirements when the terminal device quantity exceeds a certain threshold. In our experimental settings with 12 000 cloud network resources, the resource optimization achieves best performance when device quantity is 60, where the local greedy algorithm reduces the cloud cost from 5300 units to 2800 units and the GA solution only consumes 1500 units, representing the declines of 47% and 71.7%, respectively. Similar to previous simulation, for the purpose of illustrating the variance, we show the box plot for the selected local greedy scheme in Fig. 12.

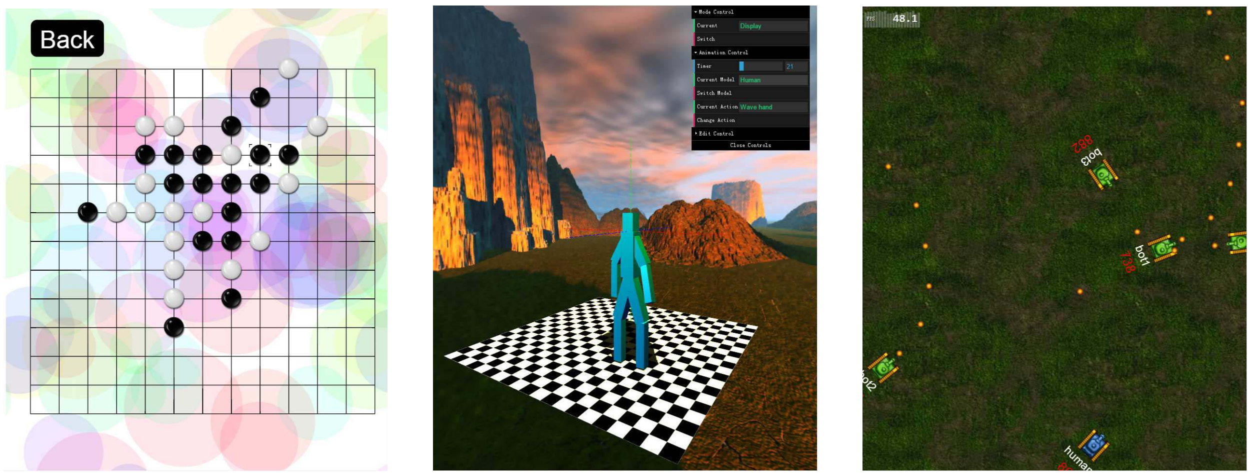
VIII. Testbed Experiments

To demonstrate the feasibility of our system design and the effectiveness of our cognitive resource optimization methodology, we have developed and deployed three prototypes in our component-based cloud gaming test bed [20]. In these three prototypes, we focus on minimizing the response delay. Runtime screenshots of the three prototypes are shown in Fig. 13 and their experimental results are discussed as follows. Note that since the component quantities in the three prototypes are relatively small, instead of applying the greedy and GA heuristic methods, the system traverses all potential partitions, i.e., performs an exhaustive search over the graph, to obtain the optimal solution.

A. Gobang Game

Gobang game is an abstract strategy board game in which players alternate in placing a chess piece of their colors on an empty intersection of the chessboard. We implement the artificial intelligence (AI) module as a component, which is feasible to migrate between cloud and players' terminals and execute on these two different environments.

Three types of devices are employed in our evaluation, including an ASUS windows 7 PC with Intel Pentium G630 @ a 2.70-GHz CPU and a 4.0-GB internal memory (RAM), an Apple iPad mini tablet with ARM Cortex-A9 CPU @ 1 GHz



(a) (b) (c)

Fig. 13. Screenshot of game prototypes on MCG. (a) Gobang game. (b) 3-D Skeletal game engine. (c) Robocode tank game.

and 512 MB RAM, and an LG G2 Android mobile phone with quad-core Snapdragon 800 CPU @ 2.26 GHz, 2.0 GB RAM, and long-term evolution (LTE) networks module. Through the public Wi-Fi network at the University of British Columbia Vancouver campus and the Fido LTE cellular data network service in Vancouver, these devices are utilized as players' terminals to access the Gobang game deployed on our developed test bed hosting in Amazon Elastic Compute Cloud (EC2).[4] By repeating the Gobang game plays with certain chess steps, we conduct the experiments with schemes iterating different combination of devices and networks, such as PC–WiFi, iPad–WiFi, Mobile-WiFi, and Mobile–LTE. For each scheme, we iterate three execution models (test bed automatic optimization, all cloud execution, and all terminal execution) and record two critical data: the AI execution time and player experienced latency. AI execution time is calculated by subtracting AI component invocation time from AI completing time, while the player experience latency is a measurement of the time difference between the time a player placing a chess piece and the time the AI placing a chess piece. As shown in Fig. 14, our cognitive engine has made correct decision: all automatic optimization solutions choose to offload AI component to the cloud, resulting in a remarkable response delay reduction from the terminal schemes.

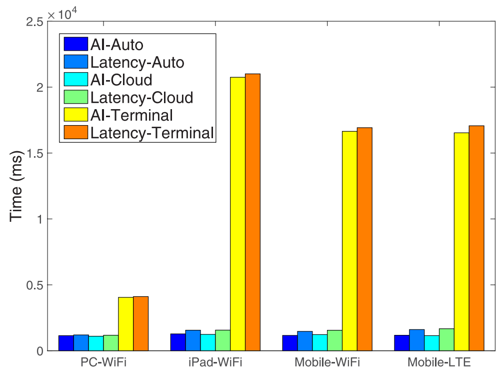


Fig. 14. Response latency comparison in the gobang game prototype.

B. 3-D Skeletal Game Engine

The 3-D skeletal engine, our second prototype, aims to challenge MCG test bed's capacity on rendering 3-D game scenes. Reference [13] has explored the possibility of partial offloading for game scene renderings. The 3-D skeletal engine is our understanding in this perspective. With a separate skinning method and forward kinematics in the OpenGL basic bone system, the implementation screenshot of a four-component prototype is shown in Fig. 13(b). To demonstrate the efficiency of cognitive optimization, we perform experiments to measure the fluency of rendered animations by the numeric value of frames per second (FPS). To explicitly control the network parameters between the cloud and terminals, we employ two identical Windows 7 computers to serve as cloud and client. On the cloud side, we installed NetBalancer[5] to control the bandwidth of NodeJS process in the cloud, for the purpose of simulating the variance of network quality in real-world cases. We design our experiments in two aspects. First, there shall be comparisons between the automatic optimization and all potential partitioning schemes. Since 3-D skeletal game engine contains four components, an iteration of their possible partitioning makes $2^4 = 16$ schemes. Therefore, we divide the total experiment time into equal 16 slides for each scheme. Second, we also concern about the engine's different performance over different network bandwidth. Hence, we repeat the experiments three times with bandwidth settings of 1000, 500, and 100 kilobytes/s, respectively.

Fig. 15 shows the results of the above experiments. The average FPS of each time slot (5 s per slot) indicates the fluency of rendered animation at the specific time period. We can conclude from the comparison between cognitive and iterations, the cognitive engine does a great job in seeking

[4]http://aws.amazon.com/ec2/

[5]https://netbalancer.com/

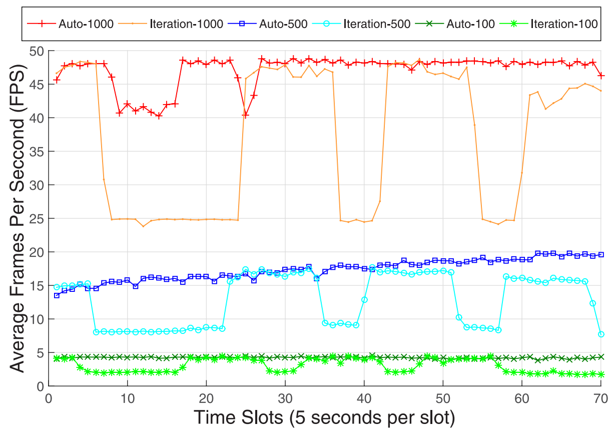


Fig. 15. Game QoS (FPS) comparison in the 3-D skeletal game engine.

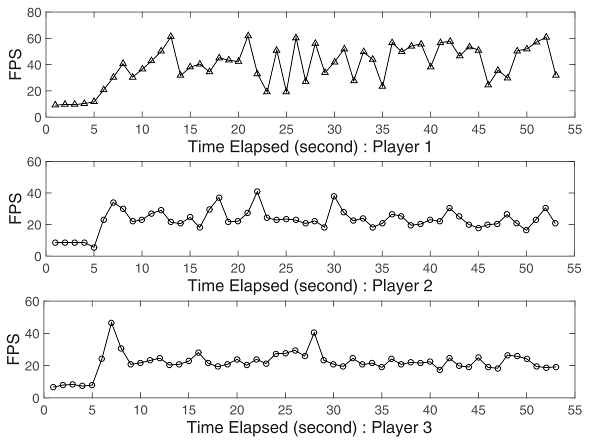


Fig. 16. Game QoS (FPS) enhancement for the Robocode tank game.

optimal partitioning solutions for the prototype: autoseries outperforms iteration series almost all the time. In addition, we derive very similar patterns from the three iteration schemes: the 1st, 5th, 6th, 8th, 9th, 11th, 12th, and 16th allocations reach the optimal FPS rate, while the rests fall to the bottom. This is a result of allocation strategy and the communication methodology between components.

C. Robocode Tank Game

The idea of the third game prototype, the Robocode tank, comes from a famous open-source educational game Robocode, which is a programming game to develop a robot battle tank to battle against other tanks in Java or .NET. The robots are controlled by competitors' AI codes and their battles are running in real time and on screen. We implement the Robocode tank game prototype in our test bed, which inherits all features of Robocode and places an additional tank controlled by players into the battlefield.

Since the Robocode tank system performance is determined by the varying complexities of tank AI, here we validate only the cognitive capacity of the MCG test bed. To record FPS traces, three players were engaged in the EC2 that hosted the tank game on the previously mentioned LG G2 Android smartphone through the Fido LTE network. Four AI-controlled tanks in the battlefield make strategy decision at a 1 s interval. As shown in Fig. 16, these players all experienced a very low FPS rate at the beginning of the gaming sessions, while the test bed eventually provided an optimal partitioning solution for them. Note that the optimal solution and solution search time for these three players are distinct from each other, according to the ever-changing network quality and game contents.

IX. Conclusion

The cognitive cloud gaming platform introduces a flexible component allocation solution that is promising cloud gaming service provision. In this paper, we have presented system modeling and the experimental results to show that the cognitive platform can provide great efficiency in terms of resource minimization and throughput optimization while guaranteeing the QoS requirements for game sessions. Our work has several limitations due to the current limitations of our test bed, which we shall address in our future work conducted over a more elaborate test bed with more sophisticated game prototypes and more concurrent devices to: 1) predict and model the bursts of component transfers and the communication costs by real-game traces; 2) develop and test more complex game prototypes with larger numbers of components; and 3) evaluate the complexity-performance tradeoffs of the proposed local greedy and GA heuristic solutions.

References

[1] P. Banerjee *et al.*, "Everything as a service: Powering the new information economy," *Computer*, vol. 44, no. 3, pp. 36–43, Mar. 2011.

[2] P. Ross, "Cloud computing's killer app: Gaming," *IEEE Spectr.*, vol. 46, no. 3, p. 14, Mar. 2009.

[3] W. Cai, M. Chen, and V. C. M. Leung, "Toward gaming as a service," *IEEE Internet Comput.*, vol. 18, no. 3, pp. 12–18, May/Jun. 2014.

[4] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "GamingAnywhere: The first open source cloud gaming system," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 10, no. 1, pp. 10:1–10:25, Jan. 2014.

[5] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proc. 19th ACM Int. Conf. Multimedia (MM)*, New York, NY, USA, 2011, pp. 1269–1272.

[6] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

[7] S. Wang and S. Dey, "Rendering adaptation to address communication and computation constraints in cloud mobile gaming," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Miami, FL, USA, Dec. 2010, pp. 1–6.

[8] S. Shi, C.-H. 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 (MM)*, New York, NY, USA, 2011, pp. 103–112.

[9] M. Hemmati, A. Javadtalab, A. A. N. Shirehjini, S. Shirmohammadi, and T. Arici, "Game as video: Bit rate reduction through adaptive object encoding," in *Proc. 23rd ACM Workshop Netw. Operating Syst. Support Digit. Audio Video (NOSSDAV)*, 2013, pp. 7–12.

[10] W. Cai, V. C. M. Leung, and L. Hu, "A cloudlet-assisted multiplayer cloud gaming system," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 144–152, Apr. 2014.

[11] R. A. Baratto, L. N. Kim, and J. Nieh, "THINC: A virtual display architecture for thin-client computing," in *Proc. 20th ACM Symp. Operating Syst. Principles (SOSP)*, New York, NY, USA, 2005, pp. 277–290.

[12] J.-M. Vanhatupa, "Browser games: The new Frontier of social gaming," in *Recent Trends in Wireless and Mobile Networks* (Communications in Computer and Information Science), vol. 84. Berlin, Germany: Springer-Verlag, 2010, pp. 349–355.

[13] D. Meilander, F. Glinka, S. Gorlatch, L. Lin, W. Zhang, and X. Liao, "Bringing mobile online games to clouds," in *Proc. 33rd IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr./May 2014, pp. 340–345.

[14] W. Cai, V. C. M. Leung, and M. Chen, "Next generation mobile cloud gaming," in *Proc. IEEE 7th Int. Symp. Service Oriented Syst. Eng. (SOSE)*, San Francisco, CA, USA, Mar. 2013, pp. 551–560.

[15] D. S. Modha, R. Ananthanarayanan, S. K. Esser, A. Ndirango, A. J. Sherbondy, and R. Singh, "Cognitive computing," *Commun. ACM*, vol. 54, no. 8, pp. 62–71, Aug. 2011.

[16] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," in *Proc. ACM/IFIP/USENIX 10th Int. Conf. Middleware (Middleware)*, Berlin, Germany, 2009, pp. 83–102.

[17] B.-G. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proc. 1st ACM Workshop Mobile Cloud Comput. Services, Social Netw. Beyond (MCS)*, New York, NY, USA, 2010, pp. 7:1–7:5.

[18] 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. (EuroSys)*, New York, NY, USA, 2011, pp. 301–314.

[19] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, New York, NY, USA, 2010, pp. 49–62.

[20] W. Cai, C. Zhou, V. C. M. Leung, and M. Chen, "A cognitive platform for mobile cloud gaming," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2013, pp. 72–79.

[21] 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. Innovative Mobile Internet Services Ubiquitous Comput. (IMIS)*, Jun./Jul. 2011, pp. 330–335.

[22] 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. IEEE/ACM NetGames*, Nov. 2012, pp. 1–6.

[23] W. Cai, X. Wang, M. Chen, and Y. Zhang, "MMOPRG traffic measurement, modeling and generator over WiFi and WiMax," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2010, pp. 1–5.

[24] S. Wang and S. Dey, "Modeling and characterizing user experience in a cloud server based mobile gaming approach," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov./Dec. 2009, pp. 1–7.

[25] 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 (NetGames)*, Nov. 2012, pp. 1–6.

[26] M. Manzano, J. A. Hernandez, M. Uruena, and E. Calle, "An empirical study of cloud gaming," in *Proc. 11th Annu. Workshop Netw. Syst. Support Games (NetGames)*, Nov. 2012, pp. 1–2.

[27] D. B. Lange and O. Mitsuru, *Programming and Deploying Java Mobile Agents Aglets*, 1st ed. Boston, MA, USA: Addison-Wesley, 1998.

[28] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users' perspective," *Math. Comput. Model.*, vol. 57, no. 11, pp. 2883–2894, Jun. 2013.

[29] W. Cai and V. C. M. Leung, "Decomposed cloud games: Design principles and challenges," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2014, pp. 1–4.

[30] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

Wei Cai (S'12) received the B.Eng. degree from Xiamen University, Xiamen, China, in 2008 and the M.Sc. degree from Seoul National University, Seoul, Korea, in 2011. He is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering, The University of British Columbia (UBC), Vancouver, BC, Canada.

He has completed visiting research with Academia Sinica, Taipei, Taiwan, The Hong Kong Polytechnic University, Hong Kong, and National Institute of Informatics, Tokyo, Japan. He has authored over ten first-author international journal/conference papers in gaming as a service, mobile cloud computing, online gaming, software engineering, and interactive multimedia.

Mr. Cai received many awards, such as the UBC Doctoral Four-Year-Fellowship, the Brain Korea 21 Scholarship, and the Excellent Student Scholarship from the Bank of China. He was a co-recipient of best paper awards from CloudCom2014, SmartComp2014, and CloudComp2013.

Henry C. B. Chan (M'98) received the B.A. and M.A. degrees from University of Cambridge, Cambridge, U.K., and the Ph.D. degree from The University of British Columbia, Vancouver, BC, Canada.

He joined The Hong Kong Polytechnic University (PolyU), Hong Kong, in 1998, where he is currently an Associate Professor with the Department of Computing. His research interests include networking/communications, cloud computing, Internet technologies, and electronic commerce.

Dr. Chan was a recipient of the IEEE Computer Society's Computer Science and Engineering Undergraduate Teaching Award for his outstanding contributions to computing education through teaching, mentoring students, and service to the education community in 2015. He received three President's awards and five faculty awards from PolyU. He was the Chair of the IEEE Hong Kong Section in 2012, and the IEEE Hong Kong Section Computer Society Chapter from 2008 to 2009.

Xiaofei Wang (S'06–M'13) received the B.S. degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2005, and the M.S. and Ph.D. degrees from the School of Computer Science and Engineering, Seoul National University, Seoul, Korea, in 2008 and 2013, respectively.

He is a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada. His research interests include social-aware multimedia service in cloud computing, cooperative backhaul caching, and traffic offloading in mobile content-centric networks.

Dr. Wang was a recipient of the Korean Government Scholarship for Excellent Foreign Students in IT Field by NIPA from 2008 to 2011, and the Global Outstanding Chinese Ph.D. Student Award in 2012.

Victor C. M. Leung (S'75–M'89–SM'97–F'03) received the B.A.Sc. (Hons.) and Ph.D. degrees in electrical engineering from The University of British Columbia (UBC), Vancouver, BC, Canada, in 1977 and 1981, respectively.

He attended the Graduate School, UBC. From 1981 to 1987, he was a Senior Member of the 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, Hong Kong. He returned to UBC as a Faculty Member in 1989, where he is currently a Professor and the TELUS Mobility Research Chair in Advanced Telecommunications Engineering with the Department of Electrical and Computer Engineering. He has co-authored over 800 technical papers in international journals and conference proceedings and 30 book chapters, and co-edited ten book titles. His research interests include wireless networks and mobile systems.

Dr. Leung is a fellow of the Royal Society of Canada, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He received the Natural Sciences and Engineering Research Council Post-Graduate Scholarship from UBC. Several of his papers had been selected for best paper awards. He was a recipient of the IEEE Vancouver Section Centennial Award and the UBC Killam Research Prize in 2012. He received the APEBC Gold Medal as the Head of the Graduating Class with the Faculty of Applied Science. He is a Registered Professional Engineer in the province of British Columbia, Canada. He was a Distinguished Lecturer of the IEEE Communications Society. He is a member of the Editorial Boards of IEEE WIRELESS COMMUNICATIONS LETTERS, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS SERIES ON GREEN COMMUNICATIONS AND NETWORKING, *Computer Communications*, and several other journals. He has served on the Editorial Boards of IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *Wireless Communications Series*, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON COMPUTERS, and *Journal of Communications and Networks*. He has guest edited many journal special issues, and provided leadership to the organizing committees and technical program committees of numerous conferences and workshops.