OPTIMIZING CLOUD GAMING SERVICE DELIVERY

by

Wei Cai

B.Eng., Xiamen University, China, 2008
M.Sc., Seoul National University, Korea, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

April 2016

Abstract

The high-profit digital gaming industry has merged with the increasing interest in transforming everything into cloud services, which leads to a novel concept called cloud gaming. In this thesis, we aim to investigate the optimization of quality of experience (QoE) for cloud gaming system, while considering different challenges, system constraints and service requirements.

First, we investigate video compression technologies based on existing cloud gaming system, in which the cloud hosts the game engine and streams rendered gaming videos to players through the Internet. We propose to cooperatively encode cloud gaming videos of different players in the same game session, in order to leverage inter-gamer redundancy. This is based on an observation that game scenes of close-by gamers have non-trivial overlapping areas, and thus adding inter-gamer predictive video frames may improve the coding efficiency. Selected games are analyzed and the trace-driven simulations demonstrate the efficiency of proposed system.

Second, we introduce a novel decomposed cloud gaming paradigm, which supports flexible migrations of gaming components between the cloud server and the players' terminals. We present the blueprint of the proposed system and discussed the cognitive resource optimization for the proposed decomposed cloud gaming system under distinct targets. This includes the minimization of cloud, network, and terminal resources and response delay, subject to QoE assurance, which is formulated as a graph partitioning problem that is solved by exhaustive searches. Extensive simulation results show the

feasibility of cognitive resource management in a cloud gaming system to efficiently adapt itself to variations in the service environments, while satisfying different QoE requirements for gaming sessions.

Finally, we explore the practical approach for the decomposed cloud gaming paradigm. We design the system framework and seek the engineering solutions for practical issues. Following these discussions, we implement the very first experimental testbed called ubiquitous cloud gaming platform. Three game prototypes are built on our testbed, which can demonstrate the feasibility and efficiency of our proposal. Experiments have been conducted to show that intelligent partitioning leads to better system performance, such as lower response latency and higher frame rate.

Preface

This thesis is based on the research I conducted under the supervision of Dr. Victor C.M. Leung. The result of this research was several articles that have been either accepted or published, or are under review. I developed the ideas for these articles and wrote them under the supervision of Dr. Leung, who also helped in revising them. For the article related to Chapter 2, Mr. Zhen Hong helped in conducting the simulations. For the conference publication of Chapter 3, Mr. Conghui Zhou and Mr. Minchen Li helped in prototype developing and Dr. Xiaofei Wang helped in addressing comments from the reviewers. For the articles related to Chapter 4, Dr. Henry C.B. Chan helped in problem formulation. In the following, the list of these publications are provided.

Publication related to Chapter 1

- Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Cheng-Hsin Hsu, Jiangchuan Liu and Victor C. M. Leung, "A Survey on Cloud Gaming: Future of Computer Games", *submitted to a peer-reviewed journal.*

- Wei Cai, Min Chen and Victor C.M. Leung, "Towards Gaming as a Service", *Internet Computing, IEEE , vol.18, no.3, pp.12-18*, May-June, 2014.

- Wei Cai, Victor C.M. Leung and Min Chen, "Next Generation Mobile Cloud Gaming", *Proceeding of IEEE 7th International Symposium on Service Oriented System Engineering (SOSE2013), pp.551-560, Hotel Sofitel, San Francisco Bay,*

USA, March 25-28, 2013.

Publication related to Chapter 2

- Wei Cai, Zhen Hong, Xiaofei Wang, Henry C.B. Chan and Victor C.M. Leung, "Quality of Experience Optimization for Cloud Gaming System with Ad-hoc Cloudlet Assistance", *IEEE Transaction on Circuits and Systems for Video Technology, vol.25, no.12, pp.1-14*, December, 2015.

- Wei Cai, Victor C.M. Leung and Long Hu, "A Cloudlet-Assisted Multiplayer Cloud Gaming System", *ACM/Springer Mobile Networks and Applications (MONET), Vol.19, No.2, pp.144-152*, April 2014.

- Wei Cai and Victor C.M. Leung, "Multiplayer Cloud Gaming System with Cooperative Video Sharing", *Proceeding of 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom2012), pp.640-645, Taipei, Taiwan*, December 3-6, 2012.

Publication related to Chapter 3

- Wei Cai, Henry C.B. Chan, Xiaofei Wang and Victor C.M. Leung, "Cognitive Resource Optimization for Decomposed Cloud Gaming Platform," *IEEE Transaction on Circuits and Systems for Video Technology, vol.25, no.12, pp.1-14*, December, 2015.

- Wei Cai, Min Chen, Conghui Zhou, Victor C.M. Leung and Henry C.B. Chan, "Resource Management for Cognitive Cloud Gaming," *Proceeding of The 2014 IEEE International Conference on Communications (ICC2014), pp.3462-3467, Sydney, Austrailia*, June 10-14, 2014

Publication related to Chapter 4

- Wei Cai, Conghui Zhou, Minchen Li, Xiuhua Li and Victor C.M. Leung, "MCG Test-bed: An Experimental Test-bed for Mobile Cloud Gaming," *Proceeding of The 13th International Conference on Mobile Systems, Applications and Services (MobiSys2015), Florence, Italy*, May 18-22, 2015.

- Wei Cai, and Victor C.M. Leung, "Decomposed Cloud Games: Design Principles and Challenges," *Proceeding of The 2014 IEEE International Conference on Multimedia and Expo (ICME2014), Chengdu, China*, July 14-18, 2014.

- Wei Cai, Conghui Zhou, Victor C.M. Leung and Min Chen, "Environment Perception for Cognitive Cloud Gaming," *Proceeding of The 4th International Conference on Cloud Computing (CloudComp2013), Wuhan, China*, Octorber 17-19, 2013.

- Wei Cai, Conghui Zhou, Victor C.M. Leung and Min Chen, "A Cognitive Platform for Mobile Cloud Gaming," *Proceeding of The IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom2013), pp.72-79, Bristol, UK*, December 2-5, 2013.

Publication related to Chapter 5

- Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Jiangchuan Liu, Victor C. M. Leung, and Cheng-Hsin Hsu, "The Future of Cloud Gaming," *Proceedings of the IEEE, vol.104, no.4, pp.687-691*, April, 2016.

Table of Contents

Appendices

List of Figures

List of Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
AMD	Advanced Micro Devices
CSS	Cascading Style Sheets
CMR-MOS	Cloud Mobile Rendering - Mean Opinion Score
CPU	Central Processing Unit
DSC	Distributed Source Coding
EaaS	Everything as a Service
EJS	Embedded JavaScript
GaaS	Gaming as a Service
GA	Genetic Algorithm
GMOS	Game Mean Opinion Score
GOP	Gourp of Pictures
GUI	Graphical User Interface
GPU	Graphics Processing Unit
GPS	Global Positioning System
IaaS	Infrastructure as a Service
IPTV	Internet Protocol television
MGUE	Mobile Game User Experience

MOS	Mean opinion score
NQoS	Network Quality of Service
MVC	Model-View-Controller
P2P	Peer-to-Peer
PaaS	Platform as a Service
PC	Personal Computer
PDA	Personal Digital Assistant
PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
QoE	Quality of Experience
RTT	Round Trip Time
SDK	Software Development Kit
SaaS	Software as a Service
UBCG	ubiquitous cloud gaming
VM	Virtual Machine
VR	Virtual Reality
WLAN	Wireless Local Area Network
WWAN	Wireless Wide Area Network

Acknowledgments

First of all, I would like to express my deep gratitude to my supervisor, Dr. Victor C.M. Leung, for providing the opportunity to study the Ph.D. program at UBC, which helped me to learn a lot about my field of study and also about myself. I also want to thank him for his invaluable support and guidance over the past years.

I would like to declare special thanks to Dr. Henry C.B. Chan and Dr. Kuan-Ta (Sheng-Wei) Chen for providing the opportunities to visit The Hong Kong Polytechnic University and Academia Sinica for a period during my research. I am grateful for their comments and advice through my research which helped in preparing my thesis.

Also, I would like to extend my appreciation to my colleagues and friends for their help and supports during the past years.

Most importantly, I feel indebted to my great family for their everlasting love and support not only during my studies at UBC but also throughout my entire life.

This work was supported by The University of British Columbia Four-Year-Fellowship and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Dedication

To my family

Chapter 1

Introduction

Over the past years, digital game has become of the most profitable products in the software marketplace and is considered the main beneficiary of digital video industry over their rival cinema. Driven by the latest designed consoles and computer games, there is an increasing population of players involved in this kind of entertainment. Moreover, total time consumed in digital games has dramatically raised, thanks to universal mobile devices and diversified mobile games. Among the most popular downloads in the application market, mobile games now dominate digital entertainment resources as a way for people to spend their idle time. According to Newzoo's Global Games Market Report, worldwide video game revenues is projected to reach $107 billion in 2017. The market becomes more engaging when the game developers start to deliver the same gaming contents through different platforms, where players are able to access identical gaming contents from different devices, at anywhere, anytime. Ideally, the controlling manner and graphical user interface (GUI) should be consistent on different platforms. We refer this kind of service paradigm as ubiquitous gaming.

The heterogeneous infrastructures of users' terminals (e.g., networks, operating system, input devices and screen size), introduce many research challenges and opportunities. One of the most critical concerns is the diversity of hardware constraints of terminals: some of them have sufficient power to render complicated three-dimensional (3D) gaming scenes (e.g., game consoles and desktop personal computers (PCs)), while others still need improvement in executing sophisticated video games for gaming

enthusiasts (e.g., smart phones, tablet and laptop). In addition, the installations of games are becoming burdens for the limited internal storage in a mobile terminal. Also, heavy battery consumption for mobile games is another significant consideration, especially when battery drain is always a big concern for smartphone users.

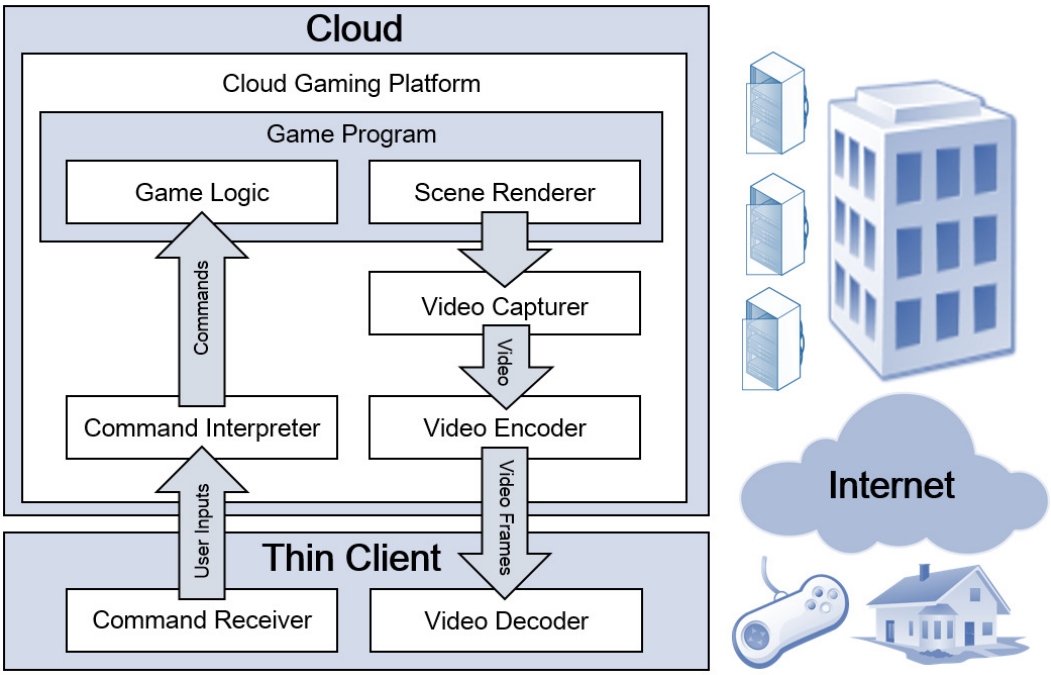


Figure 1.1: Illustration of Conventional Cloud Gaming Architecture

Realizing the cloud's virtually infinite processing power, cloud gaming becomes one of the most active research topics [1]. As an emerging software solution, it transforms traditional gaming software into Gaming as a Service (GaaS). In general, the concept of cloud gaming refers to the approach of offloading [2] game engines to the cloud server, so that the players' terminals can utilize the rich resources from the cloud to enhance their functionality. In other words, cloud gaming offers a thin-client approach for computer games by having all game data stored in the cloud's data centers and enabling computation-intensive tasks to be offloaded to the cloud. With this paradigm, it enables players to gain full access to their personalized game environments from any mobile

device and virtually anywhere. Therefore, players can overcome the intrinsic constraints of their substandard terminals, such as incompatible operating system, limited storage, insufficient computational capacity and battery drain problem. For instance, you can even play World of Warcraft on a tablet! Fig. 1.1 illustrates the conventional cloud gaming architecture.

Same as most cloud computing applications, cloud gaming services have advantages over traditional software systems. For example, *Scalability* to overcome the constraints of terminal hardware, including processing capacity, data storage and battery in mobile devices; *Ubiquitous and Cross-Platform Support* to provide immersive gaming experience; *Cost-effectiveness* for system development and software distribution; etc. Moreover, cloud gaming model exhibits more attractive features, which include: i) *Effective Anti-Piracy Solution*: Since the binary code is hosted on a secure cloud server, GaaS model is a potential solution to the long-time troubling piracy problems. In the meantime, transforming from game developing companies to game service providers is considered a more efficient and advanced business model, and as a result brings in higher profit. ii) *Click-and-Play*: Installation time is saved for players since the game copies do not need to be downloaded and set up in game terminals anymore. Nowadays, the size of gaming programs is significantly increased, yet many games are hardly, or even ever, played after they are installed [1]. As a result, "Click-and-Play" attracts more potential player's attention.

The industry has started to seize opportunities for cloud gaming. G-Cluster[2], OnLive[3], and Gaikai[4] were the most famous ones among the commercial providers. G-cluster has been building cloud gaming services since the early 2000's. In particular, G-cluster publicly

[1] http://www.gamespot.com/articles/37-percent-of-steam-games-have-never-been-played-report/1100-6419022/

[2] http://www.g-cluster.com/

[3] http://www.onlive.com/

[4] http://www.gaikai.com/

demonstrated live game streaming[5] over WiFi to a Personal Digital Assistant (PDA) in 2001, and a commercial game-on-demand service in 2004. G-cluster's service has to be tightly coupled with several third-party companies, including game developers, network operators, and game portals. This can be partially attributed to the less mature Internet connectivity and data centers, which force G-cluster to rely on network quality of service (QoS) supports from network operators. In the late 2000's, emerging cloud computing companies started offering cloud gaming services over the Internet, represented by OnLive and GaiKai. OnLive was made public in 2009 to provide a subscription based service, and hosted its servers in several states within the US, as a mean to control geographical latency. OnLive ran into financial difficulty in 2012 and ceased operations in 2015 after selling their patents to Sony. Gaikai, on the other hand, offered cloud gaming service using a different business model. They allowed users to try new games without purchasing and prompted the options for users to buy the game at the end of each gameplay. Therefore, GaiKai is more of an advertisement service to game developers for boosting sales. GaiKai was acquired by Sony in 2012, which led to a new cloud gaming service, Play Station Now, that launched in 2014.

In these commercialized cloud gaming system, video games are hosted on cloud servers and the gaming video frames are encoded by the streaming server before being transmitted over the Internet to the clients, which include interactive televisions, desktop PCs, smartphones, etc. In turns, the players' inputs are delivered to a cloud server and accepted by the game content server directly [3]. In this context, the cloud is intrinsically an interactive video generator and streaming server, while the mobile devices function as the event controller and video player that can run sophisticated games despite their restricted hardware. Nevertheless, those cloud-based video games still suffer from the bandwidth bottleneck of Internet access. The bandwidth constraints restrict the bit rate

[5] At that time, the term *cloud* is not yet well-known.

of gaming videos, while the jitter and delay affect the quality of experience (QoE) for the players. Therefore, technologies regarding real-time video rendering, compressing, and transmission quality of service (QoS) control become the most critical issues for system design [4][5]. The QoE becomes even poorer with the scenario of portable terminals with mobile networks. Even though stream-based cloud gaming service providers claim that streaming gaming videos to mobile devices can eliminate the hardware constraints of mobile devices, they also have to admit that the QoS of the existing system can not be guaranteed, since real-time video transmission requires low-latency and high-bandwidth network access. Yet, the unstable quality of the Internet infrastructure still affects all device users [6][5]. On the other hand, the capacity of mobile terminals has made remarkable enhancements, thanks to the rapid development of hardware design and technologies. Under this circumstance, another approach in providing cloud gaming services is browser game [7], which requires reliance 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, while the gaming graphics and videos are rendered by the browser, instructed by the returns from the cloud server. Compared to the normal solution of cloud-based video games, browser games leave the presentation functionality to the browsers, in order to eliminate the high bandwidth consumption for gaming video transmission. According to the study of above two types of cloud gaming, we identify that browser game is more efficient in the use of communication resources, at the expense of a heavier computation load in the user's device.

As stated in [8], the distinction between commercial video streaming and browser games are the different proportion of offloading: the former offloads everything into the cloud, while the latter only offloads the game logic.

1.1 Related Works

Many researchers from all over the world have already contributed to this topic. The first literature [1] that introduced the cloud gaming model to the academia was published in 2009, nine years after the G-cluster's demonstration of cloud gaming technology at Electronic Entertainment Expo (E3). The authors describe gaming as cloud computing's killer application and depict the blueprint of novel gaming delivery paradigm, proposed by Advanced Micro Devices (AMD), that computes a game's graphics, compress them, and sends them out over the Internet so that online gamers can run the results on platforms that are too computationally puny to render the graphics on their own. This is the most popular definition of cloud gaming adopted by most of the research work in this area. However, a recent publication [9] provides a more general definition, by envisioning a new cloud-based computer game architecture that leverages abundant and inexpensive cloud resources to support improved rendering techniques, and ensure shorter response times, better precision and higher fairness. It enables workload distribution among multiple cloud servers and game clients. Stepping forward, another work [8] explores the partitioning of the essence of cloud games into inter-dependent components, thus, defines cloud gaming as leveraging cloud resources to execute several gaming modules, thereby reducing terminal workload and increasing efficiency. According to different integration approach of cloud, the authors identify and discuss the research directions of three cloud gaming architectural frameworks: Remote Rendering, Local Rendering and Cognitive Resource Allocation.

After the successful official launch of OnLive in March 2010, the business model for cloud gaming becomes a hot topic in the research community. Riungu-kalliosaari et al. [10] conducted interview-based qualitative study to observe the dynamics related to the adoption of cloud computing within small and medium sized gaming organizations. With grounded theoretical method, the authors find that cloud gaming is relatively well-known

in the game organizations, yet they still need more clear business models and success stories to convince them to adopt cloud computing services and technology. To this end, Ojala et al. [11] started their investigations on developing business models for cloud gaming services. As a case study for software as a service (SaaS), the authors selected G-cluster, one of the most famous cloud gaming company, and analyzed its business model over five years from 2005 to 2010. They concluded that cloud gaming leads to a business model that is simpler and has fewer factors, which increases the revenue per user. In addition, they also suggested that cloud gaming could make piracy of gaming software practically impossible. Another work [12] considered the mobile cloud convergence in cloud gaming from a business model proposition. The authors discussed the first sketch of a possible business model of Kusanagi project, an end-to-end infrastructure, from domains of service, technology, organization and financial, and compared these domains with those of the three cloud examples, including G-Cluster, Gaikai and OnLive.

During the past decade of cloud gaming development, there have been several cloud gaming systems and services appearing in the market. A number of research teams survey these systems and investigate the opportunities, challenges and directions in this area. These literature, i.e., [13], [14], [15], [8], [9], [16], [17] , cover most of the commercial and academic platforms, while their concerns of open issues are greatly overlapped among the topics of response time minimization, graphical video encoding, network aware adaption, quality of experience (QoE) optimization, and cloud resource management.

Besides these common focuses, each group has particular interests and directions. The research group [13] at the University of California San Diego concentrates on developing device-aware scalable applications, which involves the open issues of extending cloud to wireless networks. Soliman et al. [14] briefly revealed related legal issues, including patents, ownership concerns, guaranteed service levels and pricing schemes. On the other hand,

piracy and hacking may be avoided, since the code is no longer delivered to the users. Wu et al. [15] explored cloud gaming architecture from the aspect of cloud computing's three layers, including Infrastructure as a Service (IaaS), SaaS and Platform as a Service (PaaS). They identify security as a potential challenge in cloud gaming, especially data protection and location. Another publication [8] examines the features of different game genres to determine their impacts on systematic design for cloud gaming service. In addition, they provided a vision on GaaS provisioning for mobile devices. Mishra et al. [9] explained how to integrate techniques from cloud and gaming research communities into a complete architecture for enhanced online gaming quality. Featured topics include the interplay between QoS and QoE metrics, game models and cloud expansion. Chen et al. [16] pointed out some unique research directions in cloud gaming, such as game integration, visualization, user interface, server selection, and resource scheduling. Chuah et al. [17] studied cloud gaming from a green media perspective. They discuss green designs of major cloud gaming subsystems: a cloud data center, graphics rendering, video compression and network delivery.

1.1.1 System Construction

The system construction for cloud gaming can be categorized into two classes: (i) *Black-box Platforms* that run unmodified games and (ii) *Augmented Platforms* that require code recompilation and/or augmentation. These two classes of cloud gaming platforms have advantages and disadvantages, and we describe representative studies in individual classes below.

The black-box platforms support unmodified games from different software developers, which reduce the cost of deploying new games, at the expense of potentially suboptimal performance. Depasquale et al.[18] presented a cloud gaming platform based

on the RemoteFX extension of Windows remote desktop protocol. Modern Windows servers leverage graphics processing unit (GPU) and Hyper-V virtual machines to enable various remote applications, including cloud games. Their experiments reveal that RemoteFX allows Windows servers to better adapt to network dynamics, but still suffers from a high frame loss rate and inferior responsiveness. Another work [19] proposes another cloud gaming platform, which consists of a distributed service platform, a distributed rendering system, and an encoding/streaming system. Their platform supports isolated audio/video capturing, multiple clients, and browser-based clients. Real experiments with 40 subjects have been done, showing high responsiveness. As the first open source black-box cloud gaming platform, GamingAnywhere [20] is designed to be extensible, portable, configurable, and open. It supports cloud servers on Windows and Linux, and its client runs on Windows, Linux, Mac OS, and Android.

On the other hand, the augmented platform [21–23] requires augmenting and recompiling existing games to leverage unique features for better gaming experience, which may potentially be time-consuming, expensive, and error-prone. For example, current games can be ported to Google's Native Client technology[6] or to Mozilla's asm.js language[7]. Several other studies focus on integrating new techniques with cloud gaming platforms for better gaming experience. Shi et al. [21] introduced a 3D image warping assisted real-time video coding method for satisfying the QoS demands of mobile cloud gaming, in terms of lower bandwidth, higher video quality, and better responsiveness. Their system selects key frames from the rendered video frames, and uses 3D image warping algorithms to interpolate the non-key frames using graphics contexts. An H.264 encoder is used to encode both key frames and residue images of non-key frames. As a case study, they integrate their system with an open source 3D tank game, and

[6] https://developer.chrome.com/native-client
[7] http://asmjs.org/

demonstrate a higher coding efficiency. Nan et al. [22] proposed a joint video and graphics streaming system for higher coding efficiency as well. Moreover, they present a rate adaptation algorithm to further minimize the bandwidth consumption. Lee et al. [23] presented a system to improve the responsiveness of mobile cloud gaming by compensating network delay. In particular, their system pre-renders potential future frames based on some prediction algorithm and delivers the rendered frames to mobile clients when the network conditions are good. These frames are then used to compensate for late video frames due to unstable networks. They integrate the proposed system with two open source games, and conduct a user study of 23 subjects. The subjects report good gaming experience under nontrivial network delay, as high as 250 ms.

1.1.2 Interaction Latency

Interaction latency is one of most important QoS factors that impacts players' experience. Claypool et al. [24] measured the contents variety of different game genres in details. 28 games from 4 perspectives, including *First-Person Linear*, *Third-Person Linear*, *Third-Person Isometric*, and *Omnipresent*, were selected to analyze their scene complexity and motion, indicated by average Intra-coded Block Size (IBS) and Percentage of Forward/backward or Intra-coded Macroblocks (PFIM), respectively. Measurements conducted by the authors suggest that Microsoft's remote desktop achieves better bitrate than NoMachine's NX client, while the NX client has a higher frame rate. The work [25] investigates OnLive's network characteristics, such as the data size and frequency being sent and the overall downlink and uplink bitrates. The authors reveal that the high downstream bitrates of OnLive games are very similar to the one in live videos; nevertheless, OnLive's upstream bitrates are much more moderate, which are comparable to traditional game upstream traffic. They also indicate that the game traffic

features are similar for three types of game genres, including *First-Person*, *Third-Person*, and *Omnipresent*, while the total bitrates can vary by as much as 50%. Another important finding is that OnLive does not demonstrate its ability in adapting bitrate and frame rates to network latency.

Chen et al. [6] analyzed a cloud gaming system's response delays and segmented it into three components: network delay, processing delay, and playout delay. With this decomposition, the authors propose a methodology to measure the latency component and apply the methodology on OnLive and StreamMyGame, two of the most popular cloud gaming platforms. The authors identify that OnLive system outperforms StreamMyGame in terms of latency, due to the different resource provisioning strategy based on game genres. A following work [26] by the same group extends the model by adding game delay, which represents the latency introduced by the game program to process commands and render the next video frame for the game scene. They also study how system design and selective parameters affect responsiveness, including scene complexity, updated region sizes, screen resolutions, and computation power. Their observation in network traffics is inline with previous work reported in [25]. Obviously, a lower network quality, including a higher packet loss rate and insufficient bandwidth, will impose a negative impact on both of OnLive and StreamMyGame, resulting lower frame rates and worse graphic quality. Moreover, by quantifying the streaming quality, the authors further reveal that OnLive implements an algorithm to adapt its frame rate to the network delay, while StreamMyGame does not.

Manzano et al. [27] collected and compared network traffic traces of OnLive and Gaikai, including packet inter-arrival times, packet size, and packet inter-departure time, to observe the difference between cloud gaming and traditional online gaming from the perspectives of network load and traffic characteristics. The authors reveal that the

packet size distributions between the two platforms are similar, while the packet inter-arrival times are distinct. Afterward, the same group published a paper [28] that claims to be the first research work on specific network protocols used by cloud gaming platforms. They focus on conducting a reverse engineering study on OnLive, based on extensive traffic traces of several games. The authors further propose a per-flow traffic model for OnLive, which can be used for network dimensioning, planning optimization and other studies.

Shea et al. [29] measured the interaction delay and image quality of the OnLive system, under diverse games, computers, and network configurations. The authors conclude that cloud procedure introduces 100 to 120 ms latency to the overall system, which requires future developments in both video encoders and streaming software. Meanwhile, the impact of compression mechanism on video quality is quite noticeable, especially under the circumstance with lower available bandwidth. These authors later presented an experimental study [30] on the performance of existing commercial games and ray-tracing applications with GPUs. According to their analysis, gaming applications in virtualized environments demonstrate poorer performance than the instances executing in non-virtualized bare-metal base-line. Detailed hardware profiling further reveals that the pass-through access introduces memory bottleneck, especially for those games with real-time interactions. Another work [31], however, has opposite observations from more advanced virtualization technologies. In the authors' measurement work, rendering with virtualized GPUs may achieve better performance than pass-through ones. In addition, if the system adopts software video coding, the central processor unit (CPU) may become the bottleneck, while hypervisor will no longer be the constraint of the system performance. Based on this analysis, the authors conclude that current virtualization techniques are already decent for cloud gaming.

Suznjevic et al. [32] measured 18 games on GamingAnywhere to analyze the correlation between the characteristics of the games played and their network traffic. The authors observe highest correlation values for motion, action game and shooter games, while the correlation values of majority of strategy games are relatively low. In contrast, for spatial metrics, the situation is reversed. They also conclude that the bandwidth usage for most games are within the range of 3 and 4 Mbit/s, except that strategy games consume fewer network resources. Another notable finding is that, gamers' action rate introduces a slight packet rate increase, but does not affect the generated network traffic volume.

Lampe et al. [33] conducted experimental evaluations of user-perceived latency in cloud games and locally executed video games. Their results, produced by a semi-automatic measurement tool called GALAMETO.KOM, indicate that cloud gaming introduces additional latency to game programs, which is approximately 85% to 800% higher than local executions. This work also highlights the significant impact of round-trip time. The measurement results confirm the hypothesis that the geographical locations of cloud data centers are important elements in determining response delay, specifically when the cloud gaming services are accessed through cellular networks.

The authors of [34] conducted a passive and an active measurement study for CloudUnion, a Chinese cloud gaming system. The authors characterize the platform from the aspects of architecture, traffic pattern, user behavior, frame rate and gaming latency. Observations include: (i) CloudUnion adopts a geo-distributed infrastructure; (ii) CloudUnion suffers from a queuing problem with different locations from time to time; (iii) the User Datagram Protocol (UDP) outperforms the Transmission Control Protocol (TCP) in terms of response delay while sacrificing the video quality; and (iv) CloudUnion adopts a conservative video rate recommendation strategy. By comparing CloudUnion and GamingAnywhere, the authors observe four common problems. First, the uplink and

downlink data rates are asymmetric. Second, low-motion games suffer from a periodical jitter at the interval of 10 seconds. Third, audio and video streams experience synchronization problems. Fourth, packet loss in network transmission degrades gaming experiences significantly.

1.1.3 Quality of Experience

Maintaining an acceptable QoE is the main criteria of the proposed cognitive platform for mobile cloud gaming. However, measuring, modeling, and predicting cloud gaming QoE are not easy tasks because QoE metrics are subjective.

Chang et al. [35] presented a measurement and modeling methodology on cloud gaming QoE using three popular remote desktop systems. Their experiment results reveal that the QoE (in gamer performance) is a function of frame rate and graphics quality, and the actual functions are derived using regression. They also show that different remote desktop systems lead to quite diverse QoE levels under the same network conditions. Jarschel et al. [36, 37] presented a testbed for a user study on cloud gaming services. Mean Opinion Score (MOS) values are used as the QoE metrics, and the resulting MOS values are found to depend on QoS parameters, such as network delay and packet loss, and contexts, such as game genres and gamer skills. Their survey also indicates that very few gamers are willing to pay a monthly fee for cloud gaming. Hence, better business models are critical for long-term success of cloud gaming. Another team [38] also conducted a subjective test in the laboratory, and considered 7 different MOS values: input sensitivity, video quality, audio quality, overall quality, complexity, pleasantness, and perceived value. They observe complex interactions among QoE metrics, QoS metrics, testbed setup, and software implementation. For example, the rate control algorithm implemented in cloud gaming client is found to interfere with the bandwidth throttled by a traffic shaper. Several open

issues are raised after analyzing the results of the user study, partially due to the limited number of participants. Slivar et al. [39] carried out a user study of in-home cloud gaming, i.e., the cloud gaming servers and clients are connected over a LAN. Several insights are revealed, e.g., switching from a standard game client to in-home cloud gaming client leads to QoE degradation, measured in MOS values. Moreover, more skilled gamers are less satisfied with in-home cloud gaming.

Some other QoE studies focus on the response delay, which is probably the most crucial performance metric in cloud gaming, where servers may be geographically far away from clients. Lee et al. [40] found that response delay imposes different levels of implications on QoE with different game genres. They also develop a model to capture this implication as a function of gamer inputs and game scene dynamics. Another group [41] makes similar conclusions after conducting extensive experiments, e.g., gamers playing action games are more sensitive to high response delay. Claypool et al. [42] performed user studies to understand the objective and subjective effects of network latency on cloud gaming. They find that both MOS values and gamer performance degrade linearly with network latency. Moreover, cloud gaming is very sensitive to network latency, similar to the traditional first-person avatar games. Raaen et al. [43] designed a user study to quantify the smallest response delay that can be detected by gamers. It is observed that some gamers can perceive less than 40 ms response delay, and half of the gamers cannot tolerate greater than 100 ms response delay.

Huang et al. [44] performed extensive cloud gaming experiments using both mobile and desktop clients. Their work reveals several interesting insights. For example, gamers are more satisfied with the graphics quality on mobile clients, while they are more satisfied with the control quality on desktop clients. Furthermore, the bitrate, frame rate, and network latency significantly affect the graphics and smoothness quality, while the control

quality only depends on the client type (mobile or desktop). Wang et al. [3, 45] built a mobile cloud gaming testbed in their laboratory for subjective tests. They propose a Game Mean Opinion Score (GMOS) model, which is a function of the game genre, streaming configuration, measured Peak Signal to Noise Ratio (PSNR), network latency, and packet loss. The derivations of model parameters are done via offline regression, and the resulting models can be used for optimizing mobile cloud gaming experience. Along this line, Liu et al. [46] proposed a Cloud Mobile Rendering - Mean Opinion Score (CMR-MOS) model, which is a variation of GMOS. CMR-MOS has been used in selecting detail levels of remote rendering applications, like cloud games.

1.1.4 Data Compression on Communications

After game scenes are computed on cloud servers, they have to be captured in proper representations and compressed before being streamed over networks. Despite the conventional real-time video compression techniques that are widely applied to video-on-demand services, compression approaches for cloud gaming can be categorized into two schemes: (i) *video compression*, which encodes two-dimensional (2D) rendered videos and potentially auxiliary videos (such as depth videos) for client side post-rendering operations, (ii) *graphics compression*, which encodes 3D structures and 2D textures, and (iii) *hybrid compression*, which combines both video and graphics compression.

Video compression utilizes graphics contexts to reduce the server transmission rate. The work [21] introduces a video encoder that selects a set of key frames in the video sequence and uses the 3D image warping coding to interpolate other non-key frames. This approach takes advantage of the pixel depth, rendering viewpoints, camera motion patterns and even the auxiliary frames that do not actually exist in the video sequence to assist

video coding. Another work [47] rectifies the camera rotation to produce video frames that are more motion estimation friendly. On client computers, the rectified videos are compensated with some camera parameters using a light-weight 2D process. In addition, a new interpolation algorithm is designed to preserve sharp edges, which are common in game scenes.

Graphics compression is proposed for better scalability, because 3D rendering is done on individual client computers. Compressing graphics data, however, is quite challenging and may consume excessive network bandwidth. Lin et al. [48] designed a cloud gaming platform based on graphics compression. Their platform has three graphics compression tools: (i) intra-frame compression, (ii) inter-frame compression, and (iii) caching. These tools are applied to graphics commands, 3D structures, and 2D textures. Another work [49] also developed a similar platform for mobile devices, where the graphics are sent from cloud servers to proxy clients, which then render game scenes for mobile devices. They also propose three graphics compression tools: (i) caching, (ii) lossy compression, and (iii) multi-layer compression. Generally speaking, tuning cloud gaming platforms based on graphics compression for heterogeneous client computers is non-trivial, because mobile (or even stationary) computers may not have enough computational power to locally render game scenes.

Hybrid Compression attempts to fully utilize the available computational power on client computers to maximize the coding efficiency. Chuah et al. [17] proposed to apply graphics compression on simplified 3D structures and 2D textures, and send them to client computers. The simplified scenes are then rendered on client computers, which is called the *base* layer. Both the full-quality video and the base-layer video are rendered on cloud servers, and the residue video is compressed using video compression and sent to client computers. This is called the *enhancement* layer. Since the base layer is compressed as

graphics and the enhancement layer is compressed as videos, the proposed approach is a hybrid scheme.

1.1.5 Adaptive Transmission

Even though data compression techniques have been applied to reduce the network transmission rate, the fluctuating network provisioning still results in unstable service quality to the players in cloud gaming system. These unpredictable factors include bandwidth, round-trip time, jitter, etc. Under this circumstance, adaptive transmission is introduced to further optimize players' QoE. It is based on common sense that, players would rather sacrifice video quality to gain smoother playing experience with poor network connections.

The first work in adaptive transmission for cloud gaming was introduced by a joint work of a Finnish research group and G-cluster in 2006 [50]. They explore the approach to adapt the gaming video transmission to available bandwidth. This is accomplished by integrating a video adaptation module into the system, which estimates the network status from network monitor in real-time and dynamically manipulates the encoding parameters, such as frame rate and quantization, to produce specific adaptive bit rate video stream. The authors utilize round trip time (RTT) jitter value to detect the network congestion, thus, decide if the bit rate adaptation should be triggered. To evaluate this proposal, the following work [51] conducted experiments on a normal television with an Internet Protocol television (IPTV) set-top-box. The authors simulated the network scenarios in home and hotels to verify that the proposed adaptation performed notably better.

Another series of investigations, conducted by a research group from University of California, San Diego, focus on the adaptation in mobile scenarios. Their first work [52] decomposed the cloud gaming system's response time into sub-components: server delay,

network uplink/downlink delay, and client delay. Among the optimization techniques applied, rate-selection algorithm provides a dynamic solution that determines the time and the way to switch the bit rate according to the network delay. As a further step, work [5] studies the potential of rendering adaptation. The authors identify the rendering parameters that affect a particular game, including realistic effect (e.g., colour depth, multi-sample, texture-filter and lighting mode), texture detail, view distance and enabling grass. Afterward, they analyze these parameters' characteristics of communications and computation costs and propose their rendering adaptation scheme, which consists of optimal adaptive rendering settings and level-selection algorithm. With the experiments conducted on commercial wireless networks, the authors demonstrate that acceptable mobile gaming user experience can be ensured by their rendering adaption technique. Thus, they claim that their proposal is able to facilitate cloud gaming over the mobile network. Subsequent works, including [46, 53], provide more solid experiments to support their claims and further extend their application scenarios to cloud mobile rendering for rich multimedia applications.

Other aspects of transmission adaptation have also been investigated in the literature. He et al. [54] considered adaptive transmission from the perspective of multi-player. The authors calculate the packet urgency based on buffer status estimation and propose a scheduling algorithm. In addition, they also suggest an adaptive video segment request scheme, which estimates media access control (MAC) queue as an additional information to determine the request time interval for each gamer, for the purpose of improving the playback experience.

Bujari et al. [55] provided a Vegas Over Access Point (VoAP) algorithm to address the flow coexistence issue in wireless cloud gaming service delivery. This research problem is introduced by the concurrent transmissions of TCP-based and UDP-based streams in the

home scenario, where the downlink requirement of gaming video exacerbates the operation of above-mentioned transport protocols. The authors' solution is to dynamically modify the advertised window, in such way the system can limit the growth of the TCP flow's sending rate.

Wu et al. [56] presented a novel transmission scheduling framework dubbed AdaPtive HFR vIdeo Streaming (APHIS) to address the issue in cloud gaming video delivery through wireless networks. The authors first propose an online video frame selection algorithm to minimize the total distortion based on network status, input video data, and delay constraint. Afterward, they introduce an unequal forward error correction (FEC) coding scheme to provide differentiated protection for Intra (I) and Predicted (P) frames with low-latency cost. The proposed APHIS framework is able to appropriately filter video frames and adjust data protection levels to optimize the quality of high frame rate (HFR) video streaming.

Hemmati et al. [57] proposed an object selection algorithm to provide an adaptive scene rendering solution. The basic idea is to exclude less important objects from the final output, thus consuming less processing time for the server to render and encode the frames. In such a way, the cloud gaming system is able to achieve a lower bit rate to stream the resulting video. The proposed algorithm evaluates the importance of objects from the game scene based on the analysis of gamers' activities and does the selection work. Experiments demonstrate that this approach reduces streaming bit rate by up to 8.8%.

1.2 Motivations

As discussed before, cloud gaming is a gaming system that leverages the cloud to enhance the service provisioning for players. To avoid confining ourselves into a specific

architectural design and technology, we investigate the essence of cloud gaming design from the perspective of a software program. Essentially, a game is a software program written in programming languages. Despite object-oriented or procedure-oriented design, we consider gaming as a loop procedure that enables the interaction between players and game logic. This particular procedure might contain different input/output (I/O) methods and involve information exchange between multiple players. However, in general, a game program is considered to be constructed by a series of modules with distinct functionality.

Fig. 1.2 illustrates a modularized game. We can see that a multi-player game consists of four main modules: the *Input Module* receives control messages from players; the *Game Logic Module* is in charge of manipulating gaming content; the *Networking Module* exchanges various information with the game server, including the interactions with other players; the *Rendering Module* renders the game video and presents to its player. In fact, if we look into the details of the *Game Logic Module*, we can further divide it into a number of components. These components invoke one another and interact with the network interface, rendering engine and I/O interface, to facilitate the gaming procedure. The set of red arrows in the figure demonstrates one of the interactions between the player and the game system during a gaming session. The player's instructions are relayed to component 5 by the *Input Module*. Afterward, the processed information is delivered to component 7, where the *Networking Module* is invoked to conduct information exchange. After the successive processes by component 6 and component 3, the *Rendering Module* generates and transmits the gaming video to the player's screen.

Based on this analysis, we classify development directions of cloud gaming system into three types:

- *Type I:* adopts a black-box approach, where unmodified games run along with cloud

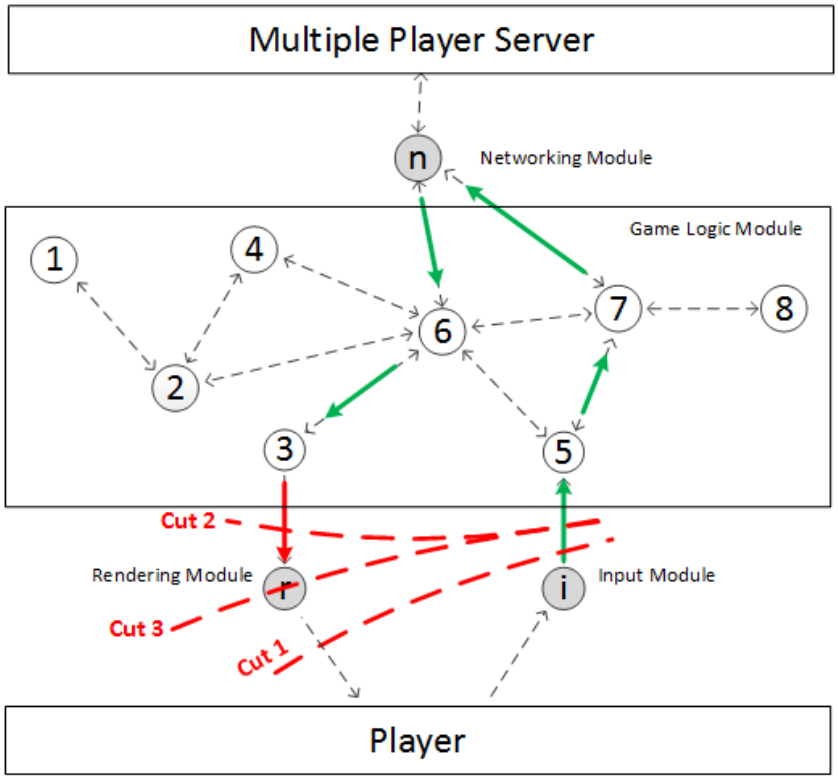


Figure 1.2: A Modularized Cloud Gaming Program

gaming servers. The rendered audio and video are captured, compressed, and streamed by cloud gaming servers to cloud gaming clients as if they are standard audio/video streams. Most commercial cloud gaming platforms take the black-box approach, probably because of its simplicity and short time-to-market. Such approach, however, leaves limited rooms for optimization. According to Fig. 1.2, Type I cloud gaming is associated with cut 1, the terminal only contains the*Input Module*, while the cloud hosts all of the remaining modules/components.

- *Type II:* refers to the cloud games that utilize terminals' computational power for graphical rendering. The rendering functions can fully or partially rely on the players' devices, which are associated with cut 2 and cut 3 in Fig. 1.2, respectively. Type II cloud gaming servers may perform more efficient data compression by using

graphical information of the game scenes, and thus significantly reduce the network transmission cost. However, this requires augmentation of game software to expose in-game contexts, such as camera location and orientation, for better interaction latency and graphics quality.

- *Type III:* dictates future programming paradigms, where games are re-written using cloud gaming Software Development Kits (SDKs). The cut between modules hosted in the player's device and those hosted in the cloud in Fig. 1.2 can be anywhere, according to the overall system optimization. While new programming paradigms offer more rooms for optimization, they are not compatible with existing games and impose steep learning curves for game developers.

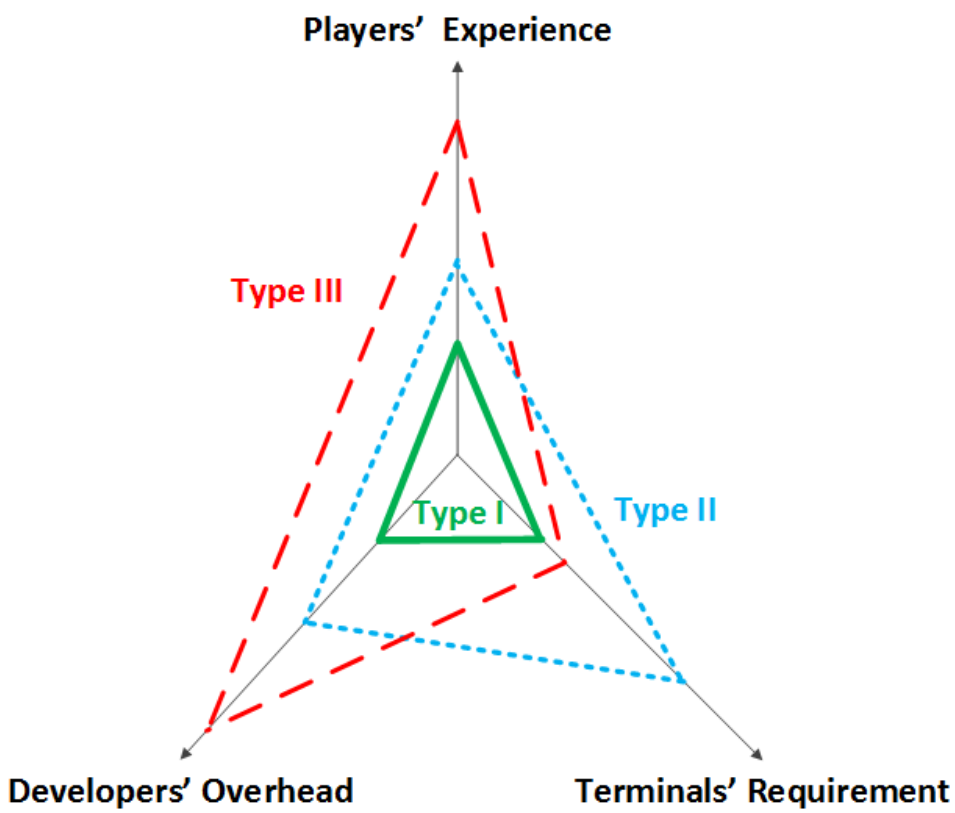


Figure 1.3: Comparisons among the three types of cloud gaming platforms

Thus far, we have briefly discussed some pros and cons of the three types of cloud gaming platforms. Fig. 1.3 compares these cloud gaming platforms from the aspects of: (i) players, (ii) developers, and (iii) terminals. For game players, the Type I approach offers almost every existing game but at lower quality. For game developers, Type III leads to higher development overhead. For terminals, Type II requires the most capable hardware, as it needs to decode graphical information and perform scene rendering locally. In general, Type III, representing the future programming paradigm approach, appears to offer more optimization opportunities at the expense of higher implementation complexity. It is, therefore, interesting to see if the gamers' demand for high gaming experience justifies the additional cost due to the implementation complexity.

We believe commercial cloud gaming services will start to implement Type II cloud gaming services, which leverage in-game contexts to either optimize the gaming experience or reduce the hardware cost. The multimedia research communities have proposed context-aware optimization algorithms for cloud gaming videos, which can be readily deployed on commercial cloud gaming platforms. Moving from Type II to Type III may happen later, due to the high implementation complexity. Web games, facilitated by centralized remote server processing and ubiquitous local browser rendering, can be considered the primary step in this direction. How soon this will happen totally depends on the development of technology and whether there are strong demands on high-fidelity cloud gaming.

Currently, these three types of cloud gaming paradigms will coexist for a period of time, as their advantages and disadvantages are complementary to each other. In this thesis, we optimize players' QoE for Type I and Type III, focusing on video encoding and software decomposition, respectively.

First, motivated by interactive multi-view streaming system [58], we consider exploring the correlations between video frames among multiple players in the same game scenario.

With the proposed cooperative video sharing system, in-game context (similar objects and background in the same scenes) are leveraged to better compress the gaming video, while the game programs running in the cloud remain unmodified.

Second, motivated by the diverse hardware capacity of players' terminals and ubiquitous gaming scenarios among different platforms, we consider a future paradigm that enables decomposition on game programs during their development. In this case, components with independent functions from one single game program can migrate between cloud and terminals, in order to dynamically adapt its service to the features of the components and the various system environment. In particular, the platform monitors the real-time environment status, sets an optimization target (e.g., the overall interaction latency, computational resource upper bound, bandwidth ceiling) and achieve the target through dynamic partitioning.

1.3 Research Questions and Challenges

Video sharing cooperative cloud gaming reduces bandwidth consumption by further compressing the video frames with additional references from peering devices. However, it also imposes several challenges and research issues. First, while the correlation between frames within single video streams have been extensively studied, the frame correlation model among different players in the same scene is still unknown. Being subject to gaming contents (e.g., the map size, avatar behavior), the similarity levels among peer players' frames will directly affect the encoding efficiency for the inter-video frames. Second, the foundation of video sharing requires an additional peer to peer (P2P) network, which supports the exchanges of complementary frames among players' terminals. Given a secondary ad hoc network as this infrastructure, the network topology and stability for engaging devices are yet to be investigated. System performance in the

presence of device mobility could also be an important issue. Third, the fast responsive nature of digital games makes real-time encoding and decoding mandatory. The latency introduced by ad hoc networks and additional cooperative decoding needs to be formulated and restricted. We address these issue in Chapter 2.

To enable decomposed cloud gaming, dynamic code partitioning is the most critical issue. Assuming the system can derive all accurate statistics from the cloud and terminal, there are still many challenging issues in making real-time partitioning decisions for running game instance. First, a prediction based on current state is necessary, as the goal is to adapt the service level to the ever changing environment. Second, the optimization targets under distinct circumstances are not always the same. How to model these targets with quantitative approaches is still unknown in this area. Third, system optimization can only be performed under the constraint that the QoE of all players are satisfied. The system, at least, needs to guarantee an acceptable QoE for most cases, which should be formulated as restrictions in all decision makings. We provide our approaches to these challenges in Chapter 3.

Decomposed cloud gaming platform is presented as the future direction of cloud gaming. However, from a practical view, the novel idea of decomposition on a game program also brings multiple research questions on implementation. First, how to conduct decomposition on a game program is still an open issue. Second, to support instant play without installation, the proposed system should support a dynamic code migration mechanism, with which the components in the cloud can be dispatched to players' terminals during the gaming sessions. Third, the invocation among components, especially remote components, requires an efficient message protocol. We still need extra efforts on developing a solution that implements global variables for decomposed functions. Fourth, the cognitive engine that makes a real-time decision for dynamic

component partitioning requires accurate and up-to-date information on system environment, terminal status and game states. An explicit implementation solution to facilitate this is yet to come. We address above issues in Chapter 4.

1.4 Contributions

In this section, we outline the contributions in this thesis to address the above-mentioned challenges. The main contributions are summarized as follows:

- In Chapter 2, we investigated a cooperative video compression technique on Type I black-box paradigm. To the best of our knowledge, this is the very first empirical work studying the correlation model of gaming videos among multiple players in the same gaming scene. Also, this is the first cooperative encoding attempt for multiplayer games. We employed the frame sharing techniques in multi-view videos to gaming applications, focusing on QoE optimizations for gamers.

- In Chapter 3, we presented a cognitive cloud gaming paradigm that balances resource utilization between cloud and terminals, facilitated by decomposed software architecture and flexible migration of gaming components. This is the first decomposition study for game programs that require extremely low response delay. Focusing on this feature, we model the inter-component relationship as a graph to further formulate the resource optimization as a graph partitioning problem. We also conducted the very first work in analyzing the feasibility of decomposition for programs.

- In Chapter 4, we investigated the principles in software decomposition and design the blueprint from the engineering perspective. We introduced a novel performance probing solution, which utilizes the dispatch procedure of mobile agent to conduct

three independent tasks together, which include collecting data from terminals to cloud, estimating the hardware capacity of terminals and measuring real-time network status. By this innovative approach, the system is capable of measuring the environmental context (e.g., network and hardware) and players' behaviors. With discussions of all practical issues and solutions, we implement the very first proposed testbed and develop three game prototypes to validate the feasibility of component based game programs and provide the first-hand experimental results.

1.5 Thesis Organization

This thesis is organized as follows. In Chapter 2, we discuss a cooperative video sharing system among multiple players in the same crowd playing the same game via a secondary ad-hoc network. The goal is to encourage exploiting the similarities of video frames among players to improve players' overall QoE. Afterward, we present the theoretical idea of a decomposed cloud gaming in Chapter 3. We provide a novel cognitive resource optimization for the decomposed cloud gaming system under diverse targets. Extensive experiments have been performed to show that intelligent partitioning leads to better system performance, such as overall latency. In Chapter 4, we discuss the challenges in developing such decomposed cloud gaming system with cognitive code migration capacity. We present a practical design methodology and implement the proposed platform. Prototype games have demonstrated the validation and efficiency of our proposal. The conclusion and some potential future work are presented in Chapter 5. Finally, the Appendices present the assumptions for the channel models and the proofs for the theorems.

Chapter 2

Ad hoc Cloudlet-Assisted Multiplayer Cloud Gaming System

2.1 Introduction

As discussed in Chapter 1, a conventional Type I cloud gaming system relies on a gaming-on-demand model. This paradigm employs a remote rendering architecture, whereby the cloud gaming service providers host their video games in cloud servers and stream the gaming video frames to the players' terminals over the Internet. In reverse, the interactions from game players are transmitted to the cloud servers over the same networks [20]. In this context, the cloud intrinsically becomes an interactive video generator and streaming server, while the users' terminals function as the event controllers and video displays. With this approach, the cloud gaming service enables the players to run sophisticated games despite the restricted hardware capacity of the terminals, at the expenses of higher costs and energy consumption in communications to access the Internet. It is obvious that video frame transmissions via the Internet can consume a huge amount of network resources, which can lead to long delays in game responses [6]. Even though there have been plenty of efforts devoted to the data compression on communications (discussed in Section 1.1.4), due to the constraints imposed by existing network infrastructure and mobile networks' charging policies, gaming-on-demand has yet to reach its promised potential.

In the same time, another trend for the game industry is the online multiplayer scenario.

Nowadays, game players are no longer satisfied with enjoying the games alone but prefer to connect to others. The interaction between players brings more challenges in the design of game servers.

In this chapter, we consider these two trends together to come up with the novel idea of improving players' QoE. We study the correlations of the gaming videos in multi-player gaming scenarios and propose an ad hoc cloudlet-assisted cloud gaming system to exploit the correlations among peer players' gaming video. Inspired by the idea of peer-to-peer sharing between multiple players in the same game, we investigated a multiplayer cloud gaming system with cooperative video sharing. Mobile devices are connected to the cloud server for real-time interactive game videos while sharing the received video frames with their peers via a secondary ad hoc network.

We consider the whole system from the following aspects: *i) Server Transmission Rate:* an intuitive approach to improving system performance is to substantially reduce the transmission rate from cloud server to the game clients in order to overcome the bottleneck of Internet access, which ensures the decode frame rate with poor Internet connections; *ii) Mobility Issue:* the proposed system assumes an ideal case in which the network bandwidth within the ad hoc cloudlet is unlimited; this is not realistic as wireless communications constrain the sharing of video frames only among mobile devices that are within a certain distance of each other, which form a time-variable group due to the mobility of terminals. *iii) Network Diversity Issue:* the previous work assumes that all mobile devices access the cloud through a common network with the same quality of service (QoS); however, the networks between individual terminals and the cloud may have different QoS due to differences in network traffic levels and channel conditions experienced by the terminals. Variations of network quality of service (NQoS) will strongly affect the overall system performance. *iv)User Quality of Experience (QoE):* the

proposed system only focuses on the optimization of server transmission rate without considering the effect of NQoS. In fact, optimizing encoding to minimize server transmissions may result in poor QoE as devices receiving larger frames from the cloud while experiencing a low network bandwidth might degrade gaming video decoding at all players' terminals. To this end, we investigate a QoE-oriented multiplayer cloud gaming system with ad-hoc cloudlet assistance.

The reminder of the chapter is organized as follows. We review related work in Section 2.2 and provide a system overview with modeling in Section 2.3. Then, we formulate the system and its QoE optimization problem in Sections 2.4 and 2.5, respectively. We propose two heuristic algorithms to obtain sub-optimal solutions with lower computational complexity in Section 2.6. Empirical experiments and trace-driven simulations, respectively, demonstrate the superior system performance compared with existing systems in Sections 2.7 and 2.8. Section 2.9 concludes the chapter.

2.2 Related Work

2.2.1 Ad Hoc Cloudlet

An ad hoc cloudlet is defined as a cooperative cloudlet formed by a group of terminals via multi-homed ad hoc network connections [59] [60] [61]. We stress that our assumption of devices being connected to multiple networks simultaneously, such as wireless wide area network (WWAN) to gaming cloud and ad hoc cloudlet to neighbor peers, is a common one in the literature [62–64] and realizable in practice (e.g., with smartphones), where different optimizations are performed exploiting the multi-homing property. It is shown in [62] that aggregation of an ad hoc group's WWAN bandwidths can speed up individual peers' infrequent but bursty content downloads just like web access. An integrated cellular

and ad hoc multicast architecture is proposed in [63], where the cellular base station delivers packets to proxy devices with good channel conditions and then the proxy devices utilize local ad hoc wireless local area network (WLAN) to relay packets to other devices. Recently, [64] utilizes a secondary ad hoc WLAN network for local recovery of WWAN broadcast/multicast packets lost during WWAN transmissions by exploiting cooperation among peers. Our proposal extends this body of work on cooperative multi-homed networks to interactive light field streaming by exploiting the correlation between requested images and content residing in peers' caches to lower server transmission rate.

2.2.2 Correlations of Videos Frames

An inter-frame, e.g., P-frame, is a frame in a video compression stream, which is expressed in terms of one or more neighbor frames. In contrast to an Intra-frame (I-frame) coding that performs compression relative to information contained only within the current frame, the "inter" part of the term refers to the use of inter-frame prediction. Its size, which affects the system performance, is subject to the correlations between the encoded video frames. The light field [65] and multiview [66] video streaming have conducted studies on this topic. The light field is a large set of spatially correlated images of the same static scene captured using a 2D array of closely spaced cameras. The correlations of light field images are studied and formulated in [67], which indicates that the correlation between two different views to a static scene is related to the geographical distances between each other. Interactive multiview video switching [58] designs a pre-encoded frame representation of a multiview sequence for a streaming server, so that streaming clients can periodically request desired views for successive video frames in time. However, compared to light field and multiview switching, the modeling of correlations between players' views is more complicated. There are infinite numbers of views as the players are adjusting their personal views while they

are walking through the scene and participating in a battlefield. The dynamic switching makes the correlation model hard to predict.

2.2.3 Real-time Video Encoding

Unlike light field and multiview switching, video encoding for cloud gaming is essentially a real-time process. The cloud encodes video frames immediately after the game scenes are rendered. The fundamental idea of encoding is very simple: it starts with an intra-coded frame, e.g., I-frame, and then followed by a certain number of interframes, such as P-frames [68], distributed source coding (DSC) frames [69], etc. Therefore, in order to achieve the desired trade-off between bit rate and error rate, how to determine the sequence of various types of frames has become one of the most critical problems in video encoding. In recent video encoding research, the GOP (Group of Pictures) [70] length is set to be adaptive, which implies a structure with one I-frame and a variable number of inter frames.

2.3 System Architecture

The architectural framework of proposed cloudlet-assisted multiplayer cloud gaming system is illustrated in Fig. 2.1. Similar to the existing cloud gaming work, instances of **Game Engine** are hosted in the cloud to provide gaming services to players. They are connected to a **Multiplayer Game Server** in conventional fashion to facilitate interactions between avatars.

The novelty of the proposed system is to introduce two additional components: **1) Video Encoder Server** is acting as a gateway, which exploits the correlations between video frames for different players to perform centralized encoding with the purpose of minimizing server transmission rate. In this work, we consider cloud as an infinite resource provider. Therefore, the computational power of the encoder server is unlimited. **2)**

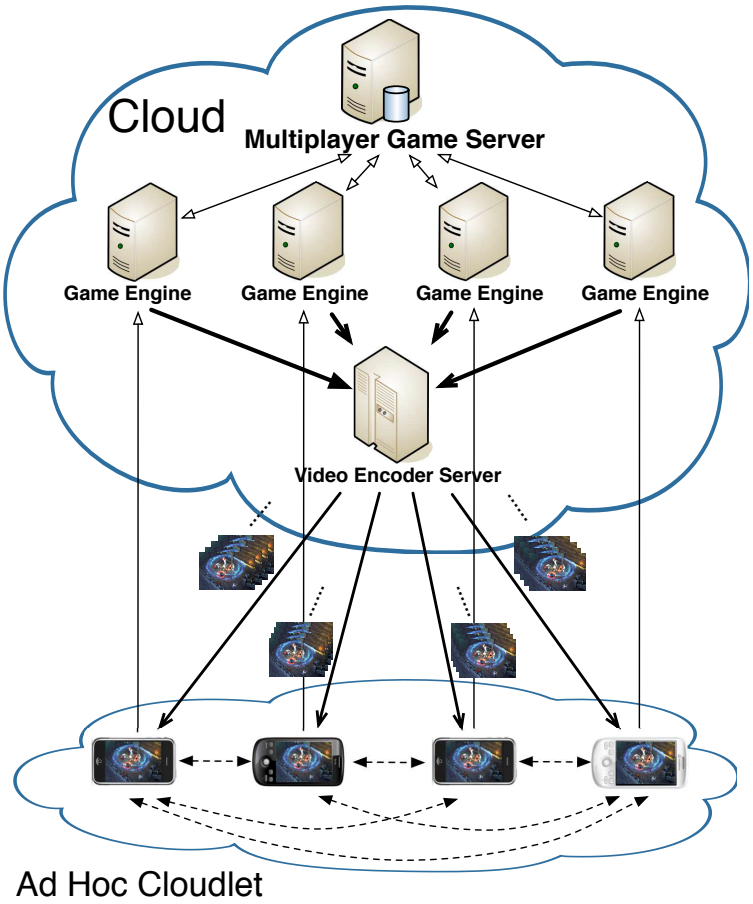


Figure 2.1: Ad Hoc Cloudlet-Assisted Multiplayer Cloud Gaming System

Ad Hoc Cloudlet is a cooperative ad hoc cloudlet constructed by the participating mobile devices. They utilize a secondary network, e.g., WiFi ad hoc network, to share the video frames they received from the cloud server. Similar to previous studies [71][72][73], we assume the network bandwidth within the ad hoc cloudlet is sufficiently large for all mobile devices in the immediate neighborhood to share their frames when needed. Thus, the bandwidth constraint inside the cloudlet will not be explicitly modeled.

2.4 System Modeling and Formulation

In this section, we model the ad-hoc cloudlet-assisted cloud gaming system, and formulate the reputation-based multiplayer fairness problem.

2.4.1 Avatar behavior Model

We choose to model the avatar behavior in a third person game, since this is the most classic and popular model, as summarized in [24]. We define the gaming map as a 2D map with an $m \times m$ screen. On this particular map, we set all avatars' initial positions to the center of the map and model the walking strategies of n players' avatars as *random walk* and *group chase* with probabilities p_{rw} and p_{gc} , respectively. This is an observation result of multiplayer games: players are randomly moving and hunting in the world (*random walk*), however, they are also prone to gather together with their teammates or opponents, in order to perform teamwork or competition (*group chase*). For *random walk* movements, an avatar either holds its position with probability p_h or moves in its adjacent n_{adj} directions with identical possibilities p_c. To simplify the model, we set $p_h = p_c = \frac{p_{rw}}{n_{adj}+1}$. For *group chase* movements, an avatar randomly selects another avatar in the scene and move towards it for a certain period of time t_{chase}. Let the probability of group chase movement be $p_{gc} = 1 - p_{rw}$, then the probabilities of any other $n-1$ target avatars to be chased will all equal to $p_{appr} = \frac{p_{gc}}{n-1}$. Note that we set the moving unit for each avatar in a unit time to be k pixels and denote w and h as the width and height of the gaming screen in pixels. Given that the system restricts the avatar to be in the center of the screen, all avatars' position coordinates will be restricted to (x, y), where $x \in [0, \frac{(m-1)w}{k}], y \in [0, \frac{(m-1)h}{k}]$.

With this model, we formulate the locations of all n avatars into two $1 \times n$ vectors X and Y, in which (X_i, Y_i) indicates the ith avatar's coordinate. With this approach, all avatar behavior models can be represented by the changes of vectors X and Y.

2.4.2 Encoding Structure

In this section, we study the frame size for two types of P-frames: *Intra-stream P-frame* and *Inter-stream P-frame*. We denote *Intra-stream P-frames* as video frames that predict to their previous frames in the same video stream and the *Inter-stream P-frames* as those predict to peer game videos' frames.

Intra-stream P-frame: Frame size of *Intra-stream P-frame* is subjected to the variance of the game video content. Apparently, if the avatar is experiencing a more dynamic scene, the P-frame size will be larger. For example, when the players are participating in a large Diablo battlefield, where many "wizards" and "demon hunters" are casting magnificent full-screen magics, the video content's amplitude of variation will be dramatic.

Inter-stream P-frame: P_{inter}, the frame size of *Inter-stream P-frame*, is subjected to the correlation between two videos for two peering game players. Before modeling P_{inter}, we need discuss the types of multiplayer games: *first-person* game, *second-person* game and *third-person* game. In video games, *first-person* refers to a graphical perspective rendered from the viewpoint of the player character, such as Counter-Strike. The *second-person* is similar to first-person but rendered from the back of the player character, which means the player can see their avatar on the screen, e.g., Grand Theft Auto. In contrast, *third-person* games provide the players a sky view, so-called God-view, to easily observe surrounding environment of the avatar and make a quick response. Classic *third-person* games include Diablo, Command & Conquer, FreeStyle, etc. For *first-person* and *second-person*, the game video is generated from the view of the avatar, where the correlation model of video frames are similar to a dynamic light field streaming. In contrast, the correlation model for *third-person* games is much simpler: the videos for peering players are very similar, or even identical, when their avatars are geographically close to each other. In fact, the

most popular multi-player games nowadays are usually designed to be *third-person*, since they provide clearer environmental information to the players. The most representative ones include League of Legend (LoL), Defense of the Ancients (DotA), and Diablo. To this end, we consider *third-person* games in this work. Since the players are all in God-view, we calculate the overlap of the two video with the avatars' coordinates, given most of the *third-person* games rolls the map with an avatar-centric manner. Fig. 2.2 demonstrates two correlated videos for player 1 and player 2.

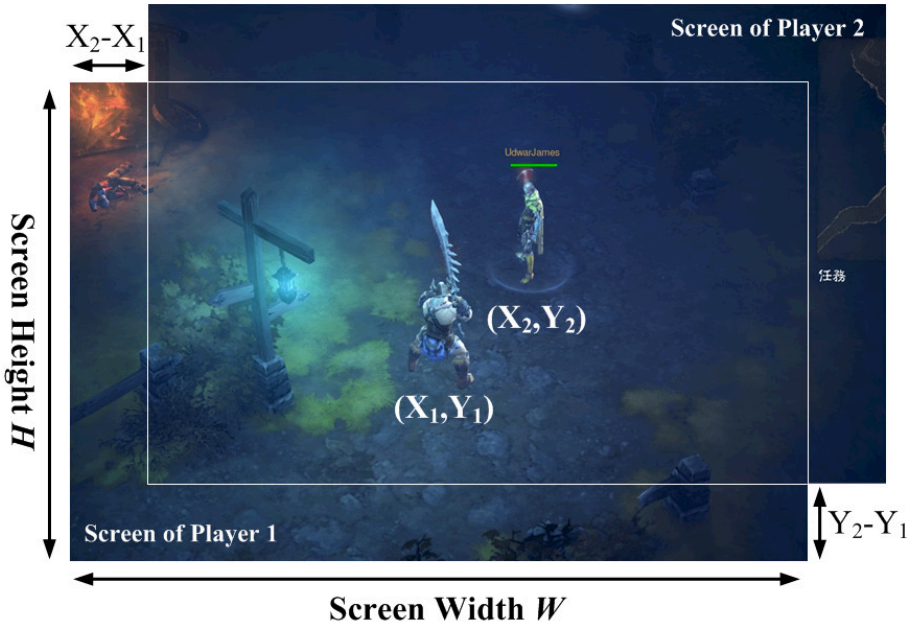


Figure 2.2: Correlation of Inter-Video Frames (Diablo III)

Fig. 2.3 shows an example of frame dependency in a four players' scenario. The squares represent video frames in video streaming, in which the paired number (t, i) denotes a frame according to the time slot t and player i. The arrows represent the frame encoding dependencies: as depicted, the video frames are able to be encoded in two dimensions. *Intra-stream P-frame* are those predicted by their previous frames in the same video stream, while the *Inter-stream P-frames* are those predicted by peer game videos' frames. For

instance, $P_{intra}[(1,2) \rightarrow (0,2)]$ indicates player 2's second Intra-stream P-frame is predicted from player 2's first decoded image, while $P_{inter}[(1,3) \rightarrow (1,2)]$ indicates player 3's second inter-stream P-frames is predicted from player 2's second decoded image.

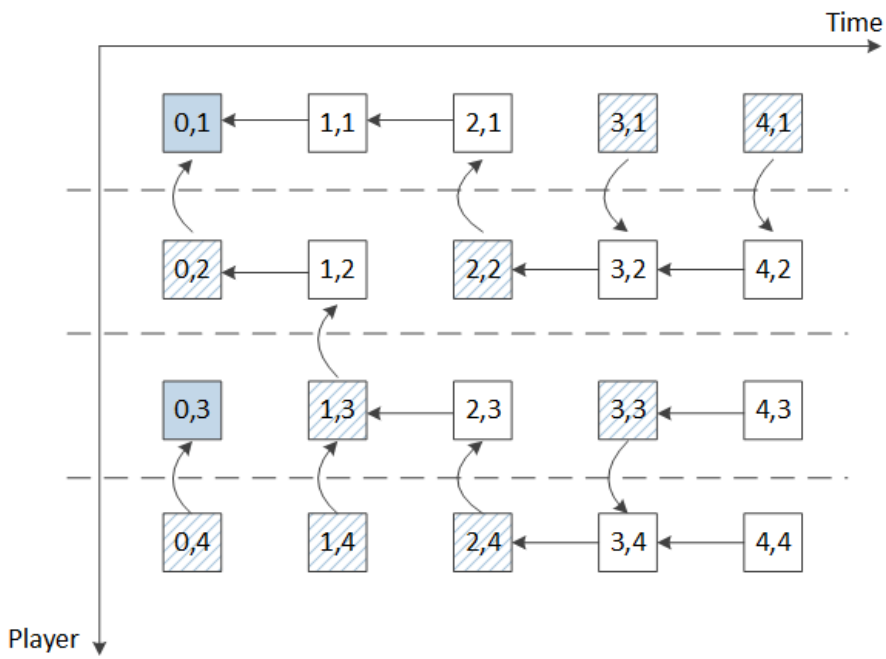


Figure 2.3: Frames Correlation in Real-time Multiplayer Game Videos

The size P_a of an *Intra-stream P-frame* is subject to the variance of the game video content. In contrast, the size P_e of an *Inter-stream P-frame* is subject to the correlation between two video streams for corresponding peering game players i and j with coordinates of (X_i, Y_i) and (X_j, Y_j), which is formulated as a function $P_e(X_i, Y_i, X_j, Y_j)$. Note that this function ensures that all avatar behavior models can be reflected in the *Inter-stream P-frame*'s size. With P_a and P_e, we define the video frame correlation matrix P. Thus, an element $P_{ij}, i \neq j$ stores the frame size of an *Inter-stream P-frame* to decode ith player's video frame by predicting jth player's. In contrast, P_{kk} saves the frame size of an *Intra-stream P-frame*, which enables the kth player to decode his/her current video by predicting the proceeding frame of his/her own video.

Noted that, to simplify the estimation of server transmission rate, we assume that the GOP length is infinite in the present work. Thus, the encoded video stream consists of a sequence of P-frames, after the first I-frame transmission. In fact, inserting I-frames to the original encoding will increase the original video frame size, which provide more opportunities for our proposed algorithm to optimize.

2.4.3 Terminal Mobility Model

To demonstrate the mobility of user terminals, we set up a square area of $r \times r\ m^2$ in which n game players are confined. The i-th player's physical position is represented as (u_i, v_i), where $i = 1, 2, ..., n$ and $u_i \in [0, r], v_i \in [0, r]$ are the X- and Y- coordinates of the location. In a typical gaming engagement, players are initially gathered in the center of the area and then randomly move within s meters in both directions every 30 seconds, i.e., $\Delta u_i \in [-s, s], \Delta v_i \in [-s, s]$ where Δu_i and Δv_i represent the movements of the ith player in X- and Y- directions in every 30 seconds. The maximum inter-device communication distance, or the device communication range, is c meters.

Based on this model, we formulate the relationship between n players' terminals as an $n \times n$ matrix E. The numeric value of an element in matrix E is defined to be either 0 or 1, where $E_{ij} = 1$ represents that the ith player's terminal is within the communication range of jth player's terminal. The matrix E is derived by:

$$E_{ij} = \begin{cases} 1, & D_{ij} \leq c \\ 0, & otherwise \end{cases} \tag{2.1}$$

where

$$D_{ij} = \sqrt{(u_i - u_j)^2 + (v_i - v_j)^2} \tag{2.2}$$

is the distance between the ith and jth players. To use the matrix E to represent the connectivity reveals the possible problem of link quality varying by dramatic changes in ad hoc network topologies. For example, in the encoding phase, the server uses E to design an encoding solution that Player 1 decodes its frame by predicting Player 2's frame. However, the connectivity between Player 1 and Player 2 is lost in the decoding phase, as the matrix E has been altered according to the terminals' mobility. In this case, Player 1's frame is not decodable. While this problem can occur, its probability can be kept sufficient low by explicitly controlling the latency between encoding and decoding of video frames to be within some acceptable levels (e.g., 200ms is "maximal tolerable" and 120ms is "hardly noticeable.", as stated in [37]). In general, the mobility of terminals will not cause dramatic network topology change in such a short time interval. Therefore, this problem is negligible.

2.4.4 Network Quality of Service Model

In practical scenarios, the players may access the cloud over different WWANs and experience different NQoS, according to the network traffic load, wireless signal strength, channel propagation condition, etc. According to a previous study [74], the players' QoE are impacted by several NQoS factors, such as network bandwidth ξ_b, network latency ξ_l, network delay variance ξ_v, and network loss rate ξ_r. These factors can be formulated into an abstract and unified *NQoS level* η as a function:

$$\eta = \mathcal{K}(\xi_b, \xi_l, \xi_v, \xi_r) \tag{2.3}$$

We can always find a function $\mathcal{K}$ that converts the NQoS parameters to an integer value η within the range $[1, n_q]$, where n_q represents the best NQoS level. According to the mobile devices' mobility and the wireless signal coverage, a mobile terminal's *NQoS level*

is expected to change from one level to another over time.

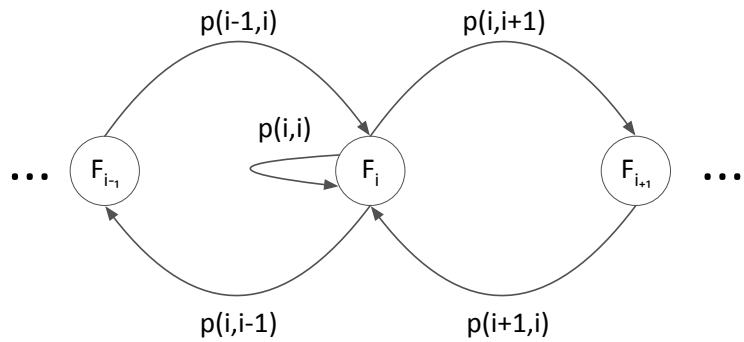


Figure 2.4: Markov Process for the Change of NQoS Level

In reality, the change of NQoS is affected by many unpredictable factors. In this work, we assume these changes are continuous, thus, the QNoS level in the next time slot depends only on the current QNoS level. Based on this assumption, we model the variations of *NQoS level* as a Markov process in which η transits from level a to b with probability $p(a, b)$, as illustrated in Fig. 2.4.

According to this model, each terminal device is associated with one *NQoS level* η, which varies from time to time. Thus, a $1 \times n$ vector U is sufficient to represent the NQoS for n players. However, to simplify our formulation and optimization in following sections, we formulate the NQoS diversity as an $n \times n$ matrix N, which is an expansion of U by $N = [U, U, ..., U]$. Note that, $\forall x, i \in [1, ..., n], N_{ix} = U_i$ represents the NQoS level η of the ith player.

2.4.5 Video Encoding Solution

For the proposed system, the most critical issue is to find an optimal video encoding solution that fulfills the requirements. In this chapter, we represent a video encoding solution by an $n \times n$ matrix M, where n represents the number of players. Each element in matrix M can take a numeric value of either 0 or 1, where $M_{ij} = 1$ represents that the ith player's

video frame is decoded by predicting to jth player's. Thus, an element $M_{kk} = 1$ implies that the kth player's terminal will download an *Intra-stream P-frame* from the cloud, while $M_{ij} = 1, i \neq j$ represents that the ith player's terminal is able to exploit the correlations from the jth player's decoded video frame, and hence its video frame is decoded by the combination of *Inter-stream P-frame* downloaded from cloud and a decoded DSC video frame *DSC-frame* received from the jth player's terminal via the ad hoc network.

To guarantee that the solution represented by a particular M can be adopted as a system encoding solution, a validation of the solution is required.

Information Integrity

In the distributed decoding system facilitated by the ad hoc cloudlet, obviously at least one player needs to download an *Intra-stream P-frame* from the cloud, in order to provide a future reference for the peer terminals. Thus, the first condition is that at least one of the diagonal elements of M is 1:

$$tr(M) = \sum_{i=1}^{n} M_{ii} \geq 1 \tag{2.4}$$

where $tr(M)$ denotes the trace of square matrix M.

Decoding Reliability

All players' terminals require either *Intra-stream P-frame* or *Inter-stream P-frame* to decode their video frames. Hence, a second condition is that the sum of elements in each row of M is equal to 1:

$$\forall i \in M_{ij}, \sum_{j=1}^{n} M_{ij} = 1 \tag{2.5}$$

Loop-Free

With the above checks, a solution is still not guaranteed to be valid. A problem called *decoding loop* can cause invalid solutions in the system. To solve this problem, we consider non-zero elements in M as the decoding vectors for n players and further represent these vectors by a graph $G = (V, E)$, where finite sets V and E contain the nodes and directed edges, respectively. Some notations are defined as follows for the analysis.

- *Node:* Each player is represented as either an *inter-node* γ_x or an *intra-node* λ_x. If a player will download images from the server via *Intra-stream P-frame*, the player is represented as an *intra-node*; otherwise, the player is represented as an *inter-node*. The set of all nodes $V = \{\gamma_1, \gamma_2, ..., \gamma_m, \lambda_1, \lambda_2, ..., \lambda_n\}$ contains m *inter-nodes* and n *intra-nodes*.

- *Edge:* Each edge connects two nodes and has a direction. An edge pointing from γ_1 to γ_2 is represented as (γ_1, γ_2). If players 1 and 2 are represented as *inter-nodes* γ_1 and γ_2, respectively, then (γ_1, γ_2) indicates that player 1 will download images from player 2 via an *Inter-stream P-frame*.

- *Path:* A finite sequence of edges connecting distinct nodes forms a path. For example, $P_x = (\gamma_1, \gamma_2, ..., \lambda_n)$ is a path with a starting point γ_1 and an end point λ_n. An isolated *intra-node* (no edge pointing in) is on a zero-path $P_y = (\lambda_n)$.

- *Loop:* A path combined with an edge pointing from the end point of the path to the starting point of the path is called a loop. For $P_x = (\gamma_1, \gamma_2, ..., \gamma_n)$, $C_y = P_x + (\gamma_n, \gamma_1)$ is a loop with n nodes $\{\gamma_1, \gamma_2, ..., \gamma_n\}$.

- *Valid Path:* A path ending at an *intra-node* is called a valid path. A zero-path is also a valid path.

- *Decoding Dependance:* An *intra-node* decodes its image by default (by downloading directly from the server, which is not shown in our analysis.) An *inter-node* decodes its image only if it is on a valid path.

- *Valid System:* If each node in the system can decode its image, then this system is a valid system.

We here make the statement that, if a graph G contains a loop, its corresponding encoding solution M is not able to be decoded in the ad hoc cloudlet. The proof is as follows:

Lemma 1 *No* intra-node *or valid path is on a loop.*

Proof Since any *intra-node* does not point to any other node, it must not be in a loop. For the same reason, since any valid path ends at an *intra-node* that is not in a loop, this path must not be in a loop.

Theorem 1 *If a system is valid, then no loop is present in the system.*

Proof Since the system is valid, each *inter-node* must be on a valid path. By Lemma 1, no *intra-node* or *inter-node* is in a loop, so no loop is present in the system.

Theorem 2 *If no loop is present in a system, the system is valid.*

Proof Suppose to the contrary that a system with no loop is not valid. Then, there exists an *inter-node* γ_x in the system such that by choosing γ_x to be the starting point, we can always find a longest path $P_x = (\gamma_x, \gamma_{x+1}, ..., \gamma_{x+n})$ with n edges. Since P_x is the longest path (i.e., γ_{x+n} cannot point to an $(n+2)th$ node) and *inter-node* γ_{x+n} has an edge pointing out, there must exist an edge $(\gamma_{x+n}, \gamma_{x+k})$ such that γ_{x+n} must point to γ_{x+k} where $k = 0, 1, 2, ..., n-1$. Then $(\gamma_{x+k}, \gamma_{x+k+1}, ..., \gamma_{x+n}) + (\gamma_{x+n}, \gamma_{x+k})$ form a loop, which is contradictory to our assumption. Therefore, the theorem has been proven.

Therefore, a system is valid if and only if no loop is present in a system.

In order to provide a loop-free solution, it is mandatory that M satisfies the following equation:

$$\forall p \in \{1, 2, ..., n\}, tr(M^p) = tr(M) \tag{2.6}$$

Proof: Please refer to Appendix A.

Mobility Constraints

The decoding procedure is constrained by the wireless connectivity in the ad hoc network, which is specified by the neighboring matrix E in our system definition.

However, in an ad hoc network, the mobile terminals may also communicate with each other over multiple hops. Denote h as the maximum number of hops allowed, we derive the h-hop neighboring matrix $\widehat{E}$ as:

$$\widehat{E_{ij}} = pos(\widetilde{E_{ij}}) = \begin{cases} 1, & \widetilde{E_{ij}} > 0 \\ 0, & otherwise \end{cases} \tag{2.7}$$

where

$$\widetilde{E} = E^h \tag{2.8}$$

Hence, we hereby formulate the mobility constraints into the validation of solution M, as shown in following equation:

$$M = M \odot \widehat{E} \tag{2.9}$$

Multi-hop Decoding Constraints

With the proposed encoder, we are able to determine the optimal solution for inter-video encoding. However, there is a practical issue to be addressed when realizing the system: since the encoder efficiently groups correlated video frames and constructs a tree to describe the dependency of the video frames, a *multi-hop decoding* might occur at the users' terminals. For example, in time slot t, given an *Intra-stream P-frame* $P_{intra}[(t,1) \rightarrow (t-1,1)]$ for player 1 and two *Inter-stream P-frames* $P_{inter}[(t,2) \rightarrow (t,1)]$ and $P_{inter}[(t,3) \rightarrow (t,2)]$ for players 2 and 3, player 3 can only decode the video frame after player 2 has decoded the frame by receiving the decoded image from player 1, which introduces larger system latency. However, gaming applications are very sensitive to latency. Therefore, *multi-hop decoding* might affect the players' QoE.

To address this issue, we provide a solution that restricts the dependency of *Inter-stream P-frame* decoding to one-hop to guarantee an acceptable level of decoding latency. This *One-Hop Inter-Stream Encoding* is enforced by introducing constraints on solution M.

Define n vector V by:

$$V_j = pos(\sum_{i=1}^{n} M_{ij}) = \begin{cases} 1, & \sum_{i=1}^{n} M_{ij} > 0 \\ 0, & otherwise \end{cases} \tag{2.10}$$

A solution M satisfies *One-Hop Inter-Stream Encoding* if and only if:

$$\forall i \in M_{ii}, M_{ii} = V_i \tag{2.11}$$

2.5 Optimization Target

2.5.1 QoE Factor

The main focus of cloud gaming system optimization is to minimize the expected server transmission rate from the cloud. Apparently, exploiting similarities from peer players' video frames with *Inter-stream P-frame* of smaller packet size is generally more advantageous in terms of data transmission speed, power consumption, data expense and others. However, the server transmission rate is not the only factor that impacts the system performance. Regarding QoE control for all players participating in this cooperative ad hoc cloudlet, there are more variables involves such as NQoS, device computing capacity, etc. For instance, with a specific encoding matrix M that minimizes transmissions, player i may be selected as the reference to which other peering terminals depend on for frame decoding, while it is struggling with poor NQoS or low computing capacities. As a consequence, its delay on video image construction introduces additional latency to the whole ad hoc cloudlet system.

To this end, we introduce a novel measure *QoE factor* θ as an index of system performance, where a smaller value of *QoE factor* θ represents a better QoE. Thanks to the recent development of hardware, most terminals have the computing power to perform complicated computations nowadays. Therefore, the *QoE factor* only considers the two most critical parameters: server transmission load and the user's NQoS. The server transmission load R is represented by an $n \times n$ matrix, where n is the number of players, with elements R_{ij} given by

$$R_{ij} = M_{ij} \cdot P_{ij} \tag{2.12}$$

where P_{ij} is defined as encoding structure stated in Section 2.4.2 and R_{ij} represents the

data size of server transmission when ith player's video frame is decoded by predicting to jth player's. We expect the value of the *QoE factor* θ to be proportional to the expected server transmission load R and inversely proportional to the NQoS level N. Hence, we may consider R and N as the *demand* and *supply* [75] of the proposed system, respectively. Referring to [75], we define

$$\theta = \mathcal{F}(R, N) \tag{2.13}$$

where R and N can be negatively correlated. In this work, without loss of generality, we choose the inverse proportional relationship between them. For instance,

$$\theta = \sum_{i=1}^{n} \sum_{j=1}^{n} (R_{ij}/N_{ij}) \tag{2.14}$$

However, it is possible to adapt the above model to various cases of the proposed system.

2.5.2 Objective Function

We can now formally define the search for the optimal encoding structure as an optimization that finds video encoding solution M, using the calculation of video correlation matrix P and NQoS N, in the feasible space Φ that possesses the smallest possible expected *QoE factor* θ, while all constraints on the encoding solution are observed. We formulate this optimization problem using the following objective function:

$$\begin{aligned} &\text{Minimize:} \quad O(M, P, N) = \sum_{i=1}^{n} \sum_{j=1}^{n} (M_{ij} \cdot P_{ij}/N_{ij}) \\ &\text{Subject to:} \quad (2.4)(2.5)(2.6)(2.9)(2.11) \end{aligned} \tag{2.15}$$

Note that, (2.11) represents the restriction of multi-hop decoding, which is optional in optimizations for different purposes.

2.6 Heuristic Approach

Given a cloud gaming session with n participants, deriving the optimal encoding solution for all terminals incurs a complexity of 2^n, which indicates a high computational cost that increases exponentially with the number of terminals accessing the cloud for gaming services. To address this issue, we investigate two heuristic approaches to support quick QoE optimization with lower computational complexity.

2.6.1 Local Greedy Approach

A local greedy approach reaches a sub-optimal encoding solution by always looking for the smallest *QoE factor* in the solution space, as shown in Algorithm 2.1. It follows the principle of a pruning algorithm to achieve the local optima.

Algorithm 2.1 Local Greedy Encoding Algorithm

```
Given correlation matrix P and NQoS matrix N
Initiate all-0 video encoding solution matrix M
Initiate search space matrix S = P/N
repeat
    for each M_mn! = 0 do
        Search smallest S_xy in S_ii and S_im
        if exist multiple smallest S_xy then
            Search smallest S_ix for each S_xy
            Select S_xy with smallest S_ix
        end if
        Set 1 → M_xy
        Set null → S_xi
    end for
until ∀i ∈ M_ij, Σ_{j=1}^{n} M_ij = 1
return M
```

According to the Algorithm 2.1, the computational complexity is reduced to n^2, which is significantly lower than 2^n. Note that if there exist multiple smallest S_{xy}, the algorithm

will further consider their potential usage as others' references to derive a relatively better encoding method.

2.6.2 One-Hop Restricted Local Greedy Approach

To solve the multi-hop decoding problem, another one-hop encoding algorithm as shown in Algorithm 2.2 is proposed in this section. The solution is basically the same as the local greedy approach but it does not consider multi-hop dependency in encoding structures.

Algorithm 2.2 One-Hop Local Greedy Encoding Algorithm

```
Given correlation matrix P and NQoS matrix N
Initiate all-0 video encoding solution matrix M
Initiate search space matrix S = P/N
repeat
  for each M_mn! = 0 and m = n do
    Search smallest S_xy in S_ii and S_im
    if exist multiple smallest S_xy and x = y then
      Search smallest S_ix for each S_xy
      Select S_xy with smallest S_ix
    else
      if exist one smallest S_xy and x = y then
        Select S_xy with smallest S_ix
      end if
    else
      if exist multiple smallest S_xy and x! = y then
        Randomly select a S_xy
      end if
    end if
    Set 1 → M_xy
    Set null → S_xi
  end for
until ∀i ∈ M_ij, Σ_{j=1}^{n} M_ij = 1
return M
```

According to Algorithm 2.2, with the same value of S_{xy}, we set up a higher priority to *Intra-stream P-frame* rather than *Inter-stream P-frame*, since an *Inter-stream P-frame*

will never be a future reference for other players with one-hop decoding restriction.

2.7 An Experimental Case Study: Diablo II

The performance of proposed system may subject to different games, since the optimization results rely on the video correlations, which is strongly related to the variance of contents and the camera positions of different players. In this work, we select Diablo II to conduct our case study, since it is considered one of the most popular and representative multiplayer action role-playing games in history. To simplify our model, two players (as a sorceress and an assassin) are connected to each other over a WLAN and start their ventures simultaneously, as illustrated in Fig. 2.5.

(a) Player 1: Sorceress (b) Player 2: Assassin

Figure 2.5: A Run-Time Screenshot for Concurrent Players in Diablo II

In order to understand the video features, we capture their gaming screen (with a resolution of 800×600) as lossless videos using Fraps[8] and encode the videos into video frames by FFmpeg[9], which is an H.264 codec widely used in online video streaming and also cloud gaming systems [20]. We set the encoding frame rate to 30 frames per second

[8]http://www.fraps.com/
[9]https://www.ffmpeg.org/

(fps), number of B-frames to 0, and leave all default parameters in FFmpeg. With infinite GOP, the frame sizes of the encoded video sequence are recorded as Intra-stream P-frame set. Note that in order to eliminate the interference from I-Frame sizes, we do not record the size of the first I-frame encoded by FFmpeg. On the other hand, we extract image sequences from the two gaming videos, and again use the FFmpeg codec with identical settings to encode Player 2's images into P-frames by predicting from player 1's concurrent image. Thus, a set of Inter-stream P-frames of Player 2 depending on Player 1 is derived. Following general research in related topics [58], we set the target video quality in terms of peak signal-to-noise ratio (PSNR) to 40dB. According to our measurements on the resulting videos, the average PSNR values for the two encoded Player 2's videos are 51.7130dB and 49.2430dB, which indicate acceptable and comparable image quality for these two encoding methods.

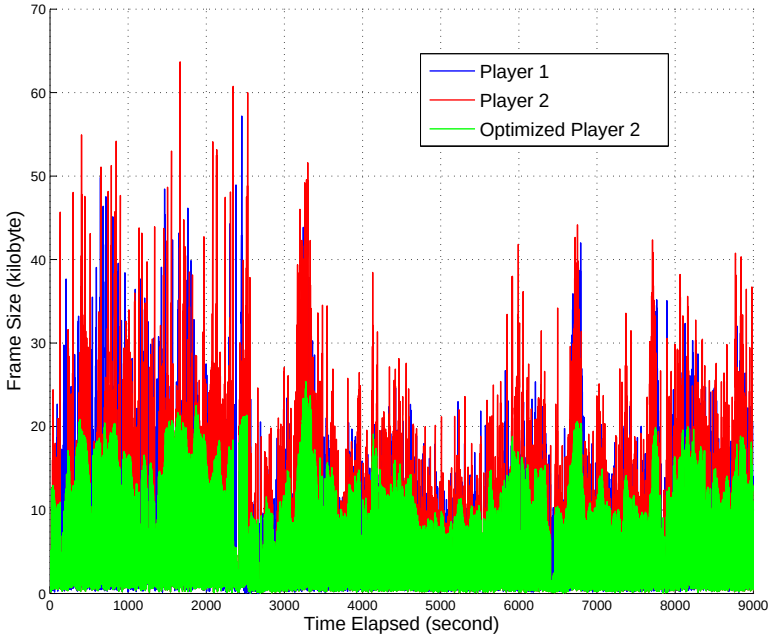


Figure 2.6: Video Frame Sizes of Diablo2 for Two Players

With the numeric number of Intra-stream and Inter-stream P-frames, it is trivial to

calculate the optimal encoding solution, given Player 2 is eligible to explore the similarity between its video sequence and Player 1's. As shown in the five-minute frame size trace in Fig. 2.6, the blue line indicates the real-time frame sizes of Player 1, while the red line indicates those of Player 2. From the figure, we can see that the two players' frame sizes fluctuate in the range of 0 to 70 kilobytes as time progresses. On the other hand, we derive the Inter-stream P-frames of Player 2 and compare their sizes to the frame size of Player 2 (red line). The smaller value between these two lines in a specific time slot is considered the optimal frame size for Player 2, which is depicted by the green line in the figure. Clearly, Player 2 can benefit greatly from exploiting the frame similarities, resulting in a significant reduction in network transmission by 23.26% (from 111906600 bytes to 85871751 bytes). Moreover, it is also remarkable that the optimized frame sizes are smoother, in comparison to original Intra-stream P-frame. This will enhance the QoE for cloud gaming participants, especially when network bandwidth is scarce.

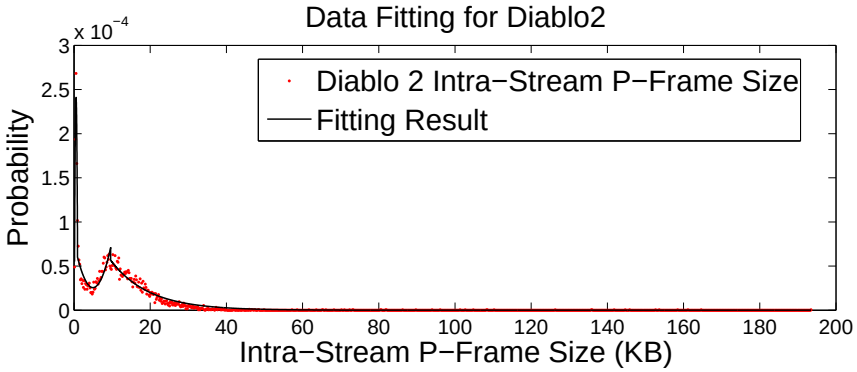
2.8 Trace-Driven Simulations

In this section, we conduct trace-driven simulations to evaluate the performance of the cloud gaming system and our proposed algorithms. We first study the size of P-frames in real scenarios and then simulate gaming sessions to evaluate the QoE optimization for the proposed system.

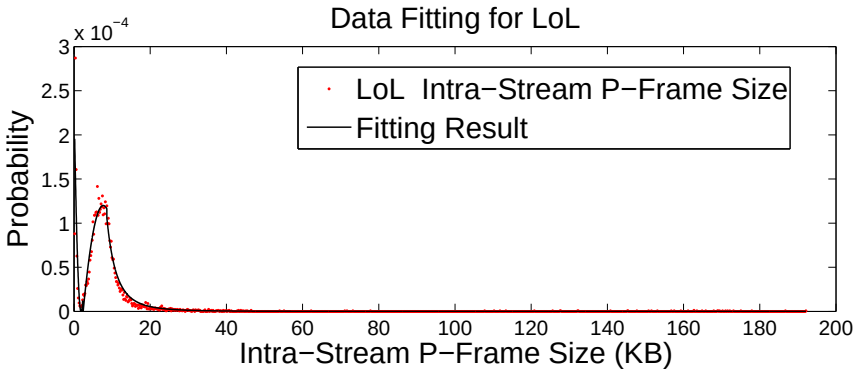
2.8.1 Study on Intra-Stream P-Frame Size

To obtain empirical models of Intra-Stream P-Frame size, we conduct video measurement experiments for three different games: Diablo II, League of Legend (LoL) and NBA2K14. We played several sessions of these games and captured their videos for analysis. To simply the comparison, we set the screen resolution of Diablo II and NBA2K14 to 800×600, while

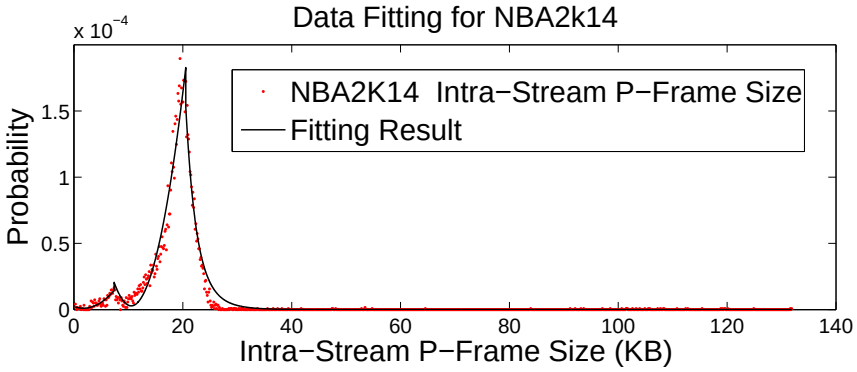
the LoL screen is set to 1024×768, which is the lowest game mode. After H.264 encoding with FFmpeg, we obtain the Probability Distribution Function (PDF) of the frame sizes for these games, as depicted in Fig.2.7.



(a) Intra-Stream P-frame Size for Diablo II



(b) Intra-Stream P-frame Size for LoL



(c) Intra-Stream P-frame Size for NBA2K14

Figure 2.7: An Illustration of Distributions of Inter-stream P-frame Sizes

As we can see from the figures, the PDFs of video frames demonstrate different patterns from game to game. This implies that we cannot represent all types of games with a single model. To support our subsequent simulations, we conduct curve-fitting for Diablo II to obtain the following piecewise function:

$$f(x) = \begin{cases} a_1x^2 + b_1x + c_1, & 2.03 * 10^{-3} \leq x \leq 0.91 \\ a_2x^2 + b_2x + c_2, & 0.91 \leq x \leq 9.62 \\ a_3e^{b_3x}, & 9.62 \leq x \leq 193.36 \end{cases} \tag{2.16}$$

where $a_1 = -1.136 * 10^{-3}, b_1 = 1.234 * 10^{-3},_1 = -9.428 * 10^{-5}, a_2 = 2.101 * 10^{-6}, b_2 = -2.092 * 10^{-5}, c_2 = 7.765 * 10^{-5}, a_3 = 1.907 * 10^{-4}, b_3 = -0.1178$. The three R-squared values for the goodness of fit are $R_1^2 = 0.8968$, $R_2^2 = 0.6013$ and $R_3^2 = 0.9437$. Based on this equation, we create a random frame size generator $x = f^{-1}(y)$ to simulate the variety of Intra-stream P-frames.

2.8.2 Study on Inter-Stream P-Frame Size

As stated in our system model, we assume that the distances between avatars are positively correlated to the frame sizes of *Inter-stream P-Frames*. In this section, we conduct a two-player experiment based on Diablo II to verify this assumption and quantitatively measure their relationships.

The experiment was conducted as follows. An amazon (Player 1) stands still in a battlefield and a necromancer (Player 2) walks around the amazon at a certain pace, from near toward far. For each move, both players capture screenshots for future comparison. With 26 collected screenshots, we encoded Player 1's images as I-frames, while Player 2's images were encoded as *Inter-stream P-Frames* predicted from the corresponding I-frames. The data sizes of these *Inter-stream P-Frames* are depicted in Fig. 2.8.

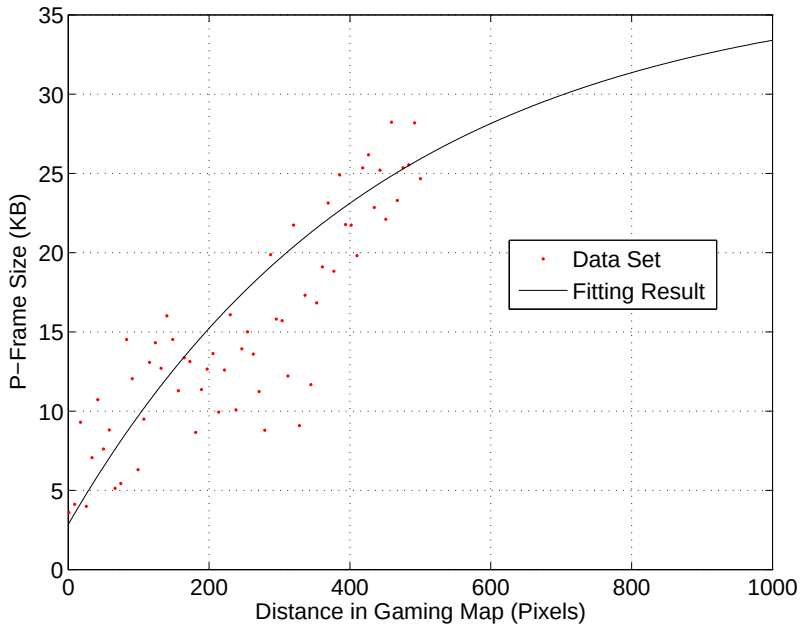


Figure 2.8: Exponential Fitting of Inter-stream P-frame Size

Based on these experimental data, we investigate the relationship between frame size and viewing distance by curve-fitting the P-frames sizes as follows.

$$f(x) = a(1 - b \times e^{-\frac{x}{c}}) \tag{2.17}$$

where $a = 37$, $b = 0.9231$, $c = 13.89$ and the R-squared for goodness of fit is 0.731. With this approach, we derive the function of $P_e()$ as follows:

$$P_e(X_i, Y_i, X_j, Y_j) = a * (1 - b * e^{\frac{-\sqrt{(X_i - Y_j)^2 + (X_i - Y_j)^2}}{c}}) \tag{2.18}$$

Fig. 2.9 illustrates the function of $P_e()$, in which the size of *Inter-stream P-frame* is directly associated with the gaming scene's coordinate system.

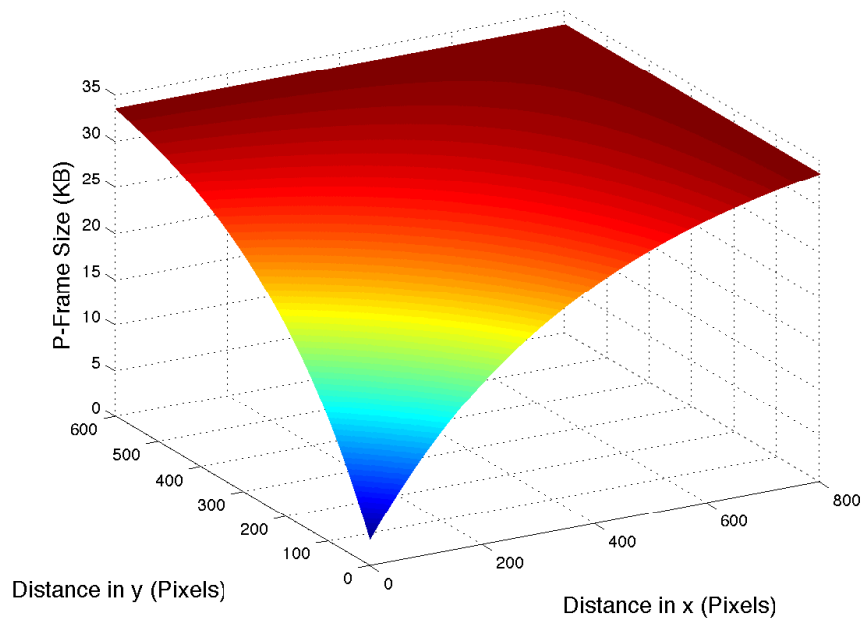


Figure 2.9: Inter-stream P-frame Size Defined by P(e)

2.8.3 Experimental Setup

To validate the performance of our proposed system and encoding schemes, we set up the following experiments. The expectation of Intra-stream P-frame μ comes from the average value of P-frame size in the Stanford Bunny light field. Since our previous work [76] has performed studies on avatars' interactional model, this chapter focuses on the other parameters that impacts the QoE. To emulate the network coverage and wireless channel in real world, we simplify the model in simulations by enforcing state transitions of NQoS to neighboring levels as follows:

$$\eta(i) = \begin{cases} i, & p(i,i) \\ i+1, & p(i,i+1) \\ i-1, & p(i,i-1) \end{cases} \tag{2.19}$$

In our experiments, we initiate all players' devices with random values of NQoS level following a uniform distribution and renew these values by the above equation in every 20 seconds, according to the movement interval of players. In each iteration of the simulation, we assign the terminals with different random values of Markov chain transition probabilities $p(i,i), p(i,i+1), p(i,i-1)$, which represent their distinct NQoS transition. Other default values of the simulation parameters are shown in Table 2.1.

Table 2.1: Default Simulation Parameters

number of players n	8
player gaming region r	100 meters
player movement unit s	5 meters
best NQoS level n_q	10
terminal maximal allowed hops h	1
terminal maximal communication range c	25 meters
simulation time T	600 seconds
fps (frame per second)	25
map size $m \times m$	12
screen width w	800 pixels
screen height h	600 pixels
unit step pixels k	32 pixels
adjunction direction n_{adj}	8
random walk probability p_{rw}	0.4
chase time t_{chase}	5 seconds

All experiments are conducted with random seeds. For comparisons, we derive the average QoE factors of five schemes: *Intra-only* represents the traditional encoding method without ad hoc cloudlet assistance, *Optimal* represents the global optimal QoE with our optimization, *Optimal (One-hop)* represents the global optimal QoE with one-hop encoding restriction, while *Greedy* and *Greedy (One-hop)* represents the simulation results derived by our proposed greedy algorithms with and without one-hop encoding restriction.

2.8.4 Effects of Number of Players

We first evaluate the effects of the number of players on QoE Performance. With a fixed size gaming map and avatar behavior model, more game participants will increase the chance of correlated videos generated in the cloud server that exploits better encoding schemes with the help of *Inter-stream P-frames*. Since expected server transmission rate is proportional to our defined QoE factor, it is obvious that lower expected server transmission rate results in smaller QoE factor.

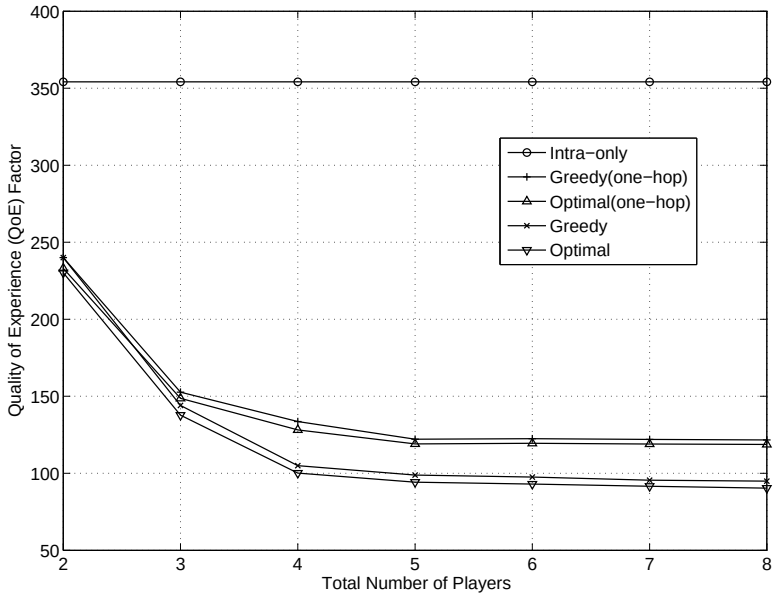


Figure 2.10: Effects of Number of Players on QoE

As illustrated in Fig. 2.10, the values of the QoE factor for *Intra-only* remain unchanged as the number of game players increases, while those of the other 4 schemes keep decreasing. The *Optimal* solution exhibits the most improvement on QoE, dropping from 230 to 100 when the number of players increases from 2 to 8. As a sub-optimal solution, the *Greedy* algorithm also demonstrates a great performance by underperforming the *Optimal* algorithm by only around 3%. With the one-hop decoding

restriction, *Optimal (One-hop)* encoding reduces the values of the QoE factor by 27.3% to 65.9% compared to *Intra-only*. Again, *Greedy (One-hop)* only slightly underperforms the corresponding optimal scheme, reducing the QoE factor by up to 65.6% compared to *Intra-only*. Another interesting phenomenon shown by this figure is that, as the number of players increases, the QoE factor of *Optimal* decreases more significantly than *Optimal (One-hop)* does. The reason is that, when more players are collaboration over the cloudlet, multiple hop correlations between gaming videos are more likely to occur, which benefits the *Optimal* scheme but not the one-hop restricted scheme.

2.8.5 Effects of Maximum Communication Range between Terminals

The terminals' connectivity to each other is an important factor that affects the packet exchange in the ad hoc cloudlet, which impacts the cooperative *Inter-stream P-frame* sharing in QoE optimization. In this section, we evaluate the five schemes over different maximum communication ranges between the terminals', ranging from 0 to 80.

As illustrated in Fig. 2.11, the values of QoE factor for *Intra-only* are not affected by changes in the maximum communication range, while the QoE factors of the other 4 schemes with ad hoc cloudlet assistance keep reducing along with the increase of communication range from 0 to 50 meters. Note that, the QoE factor of the *Optimal* scheme drops from 480 to 170 when the communication range is increased from 0 to 25 meters, which is a 72.9% improvement in QoE. In fact, at a communication range of 25 meters, even the *Greedy(one-hop)* solution can achieve a 64.6% reduction in the QoE factor compared to the *Intra-only* scheme. However, increasing the maximum communication range to longer than 50 meters does not yield further QoE improvements.

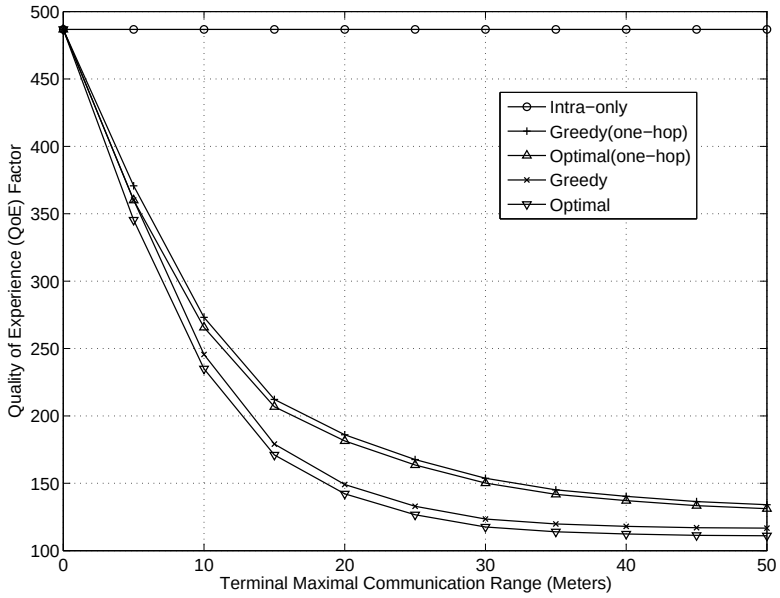


Figure 2.11: Effects of Maximum Communication Range of Terminals on QoE

2.8.6 Effects of Maximum Number of Hops Allowed

Another factor that impacts the efficiency of the ad hoc cloudlet is the maximum number of hops allowed. As discussed in Section 2.4.5, the multi-hop packet relay in ad hoc networks is a key element of constructing the neighbor matrix E, which determines if a *Inter-stream P-frame* sharing between terminals is possible. Allowing more hops for terminals to exchange video frames increases the chances of exploiting correlations between peering devices.

Fig. 2.12 compares the performance of the 5 schemes by simulations, in which the maximum number of hops allowed is varied from 0 to 6. Intuitively, allowing more hops in the ad hoc cloudlet should benefit the QoE optimization. However, in our experimental settings, these improvements are not significant, especially when we further consider the latencies introduced by multi-hop relays.

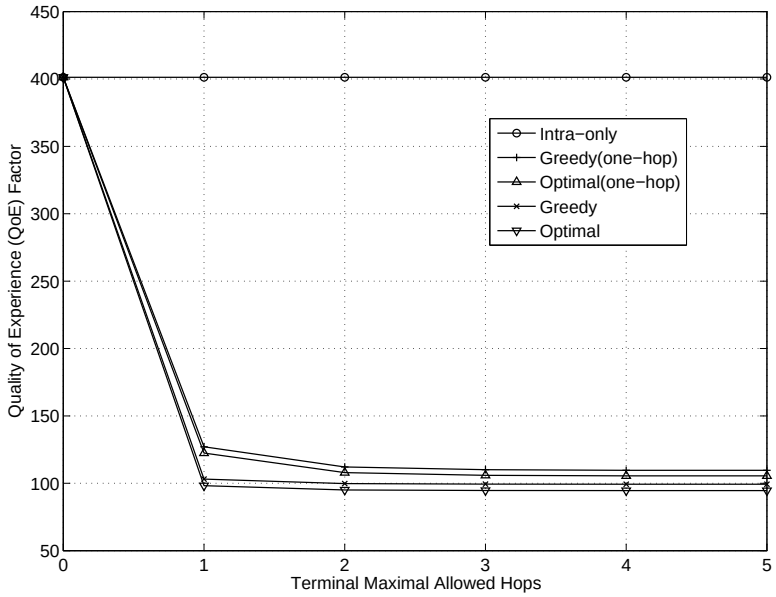


Figure 2.12: Effects of Maximum Number of Hops Allowed on QoE

2.8.7 Effects of NQoS

As discussed in Section 2.4.4, the variation of NQoS is a critical element in QoE optimization. Avoiding the download of large frames from devices with poor NQoS could benefit all game participants in the cloud gaming system. In this section, we enumerate combinations of $p(i, i+1)$ and $p(i, i-1)$ settings in the range of $[0, 0.5]$ to evaluate the effects of NQoS.

As shown in Fig. 2.13, both *Intra-only* and *Optimal* reach a peak value at $p(i, i+1) = 0, p(i, i-1) = 0.5$, since this setting makes all terminals' NQoS to become worse, which substantially degrades the overall QoE. Note that with the same NQoS parameters, the proposed *Optimal* solution outperforms the conventional *Intra-only* scheme with lower values of QoE factor. More importantly, the QoE improvement of our proposed ad hoc cloudlet system is especially pronounced in the scenarios where NQoS is poor; it yields a peak QoE factor of around 1000, which is a 40% of the result for the *Intra-only* solution.

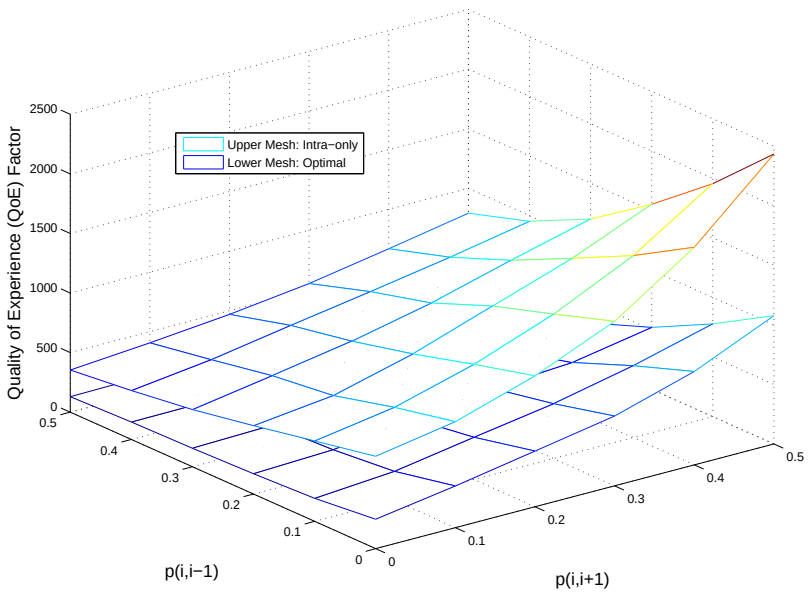


Figure 2.13: Effects of NQoS on QoE

2.9 Summary

In recent years, cooperative video sharing with the support of ad hoc cloudlets has been investigated as a bandwidth-efficient solution in streaming-based cloud gaming systems. In this work, we have improved the system model in our previous work by considering terminal mobility and the variations of NQoE. We have formulated the QoE-oriented optimization problem, with and without multi-hop decoding restriction. Heuristic solutions have also been proposed to reduce computational complexity and enable quick encoding in real-time gaming video streaming. Results from empirical studies on Diablo II and extensive gaming trace-drive simulations have been presented to demonstrate the effectiveness of the proposed QoE optimization scheme and the efficiency of the heuristic algorithms.

Chapter 3

Cognitive Cloud Gaming Platform

3.1 Introduction

As stated in Section 1.2 of Chapter 1, the main distinction between the two existing cloud-based gaming models is the proportion of offloading. However, both of them are still of insufficient flexibility, given the various scenarios of playing cloud games on terminals. 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. In this chapter, we present a cognitive and flexible cloud gaming platform, which learns the player's environment (i.e., the combination of terminal and access network) and adapts the cloud gaming service to these evaluations. This kind of platform decomposes the game program into inter-dependent components that can be distributed to either the cloud or local terminal for execution, which achieves flexible resource allocation. Accordingly, the adaptive, interactive, contextual, iterative and stateful paradigm of cognitive computing [77] has motivated us to build a context-aware scheme that learns players' behavior 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 quality of service (QoS) objectives of the gaming sessions. The remaining sections of the chapter are organized as follows. We review related work in Section 3.2 and propose the framework in Section 3.3. Then, in Section 3.4, we model and formulate the decomposed cloud games

as a graph partitioning problem and perform theoretical analysis on optimal solutions. QoS requirements and the optimization targets for cognitive cloud resource management are described in Section 3.5. We further suggest two heuristic approaches for scalable implementations, based on local greedy and genetic algorithm (GA) approaches, in Section 3.6. Results of simulations are presented in Sections 3.7. Section 3.8 summarizes the chapter.

3.2 Related Work

The resource allocation optimization problem considered in this chapter 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. The work [78] first introduced a K-step algorithm to compute partitioning on-the-fly, when a phone connects to the server, and specified its resources and requirements. Furthermore, the work [79] formulated the dynamic partitioning problem and discussed the supporting platform that facilitates it. A dynamic partitioning system named CloneCloud was designed in [80]. 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 [81], 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 JavaScript language.

3.3 System Overview

Fig. 3.1 illustrates a two-layer conceptual framework that facilitates the dynamic partitioning.

3.3.1 Conceptual Framework

The first layer is designed for the component-based games. These game components are able to migrate from the cloud to the mobile terminal via the 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

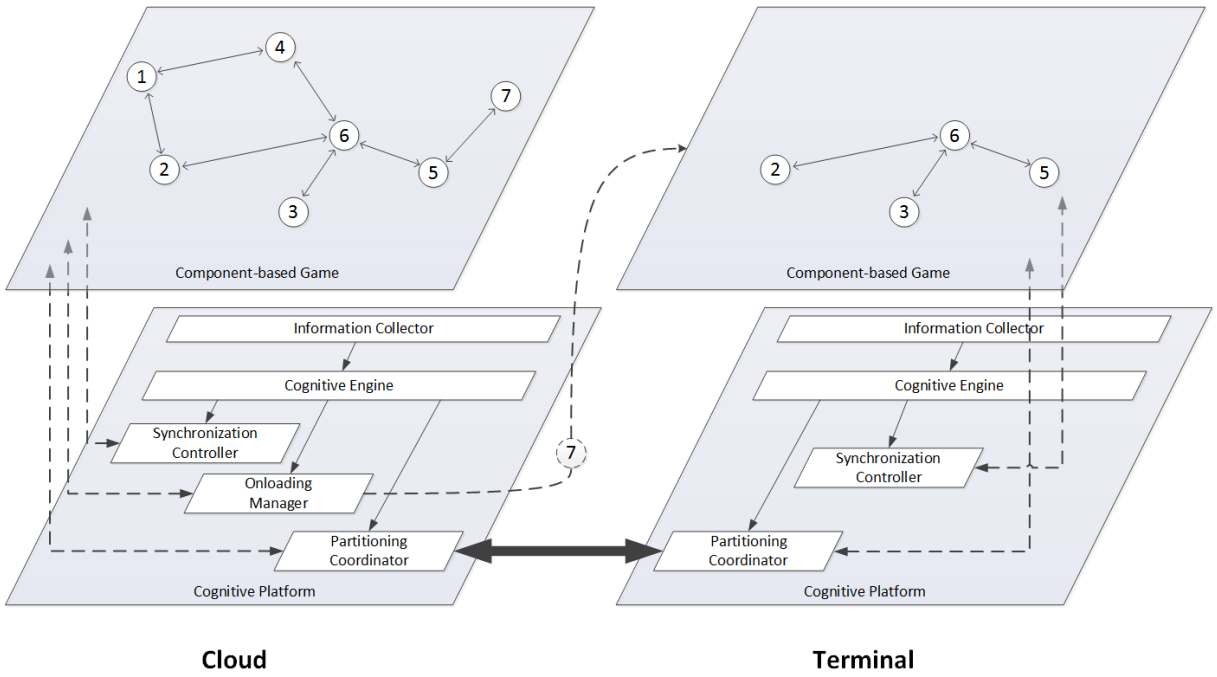


Figure 3.1: Cognitive Cloud Gaming Platform

distributed in the cloud and mobile terminals. Note that the *Onloading Manager*, *Partitioning Coordinator* and *Synchronization Controller* work with the *Cognitive Engine* for the purpose of maintaining an acceptable QoS for players.

As an infrastructure, the cognitive platform collects information from cloud, terminal and networks. Afterward, this information is forwarded to *Cognitive Engine*s hosted in both cloud and terminal sides. Of course, these two engines might have different roles: the one in the terminal acts as the first filter for data, while the one in the cloud can perform sophisticated analysis that supports decision making of dynamic partitioning. Nevertheless, both of them are imperative for the system.

3.3.2 Cognitive Resource Management

In this section, we provide an overview of the proposed cognitive resource management. As a cognitive system, it is cognitive of resources and characteristics of the cloud, the access

network, and the end-user devices, to enable dynamic utilization of these resources. To the best of our knowledge, a QoE-oriented cognitive system is not yet investigated for cloud gaming systems.

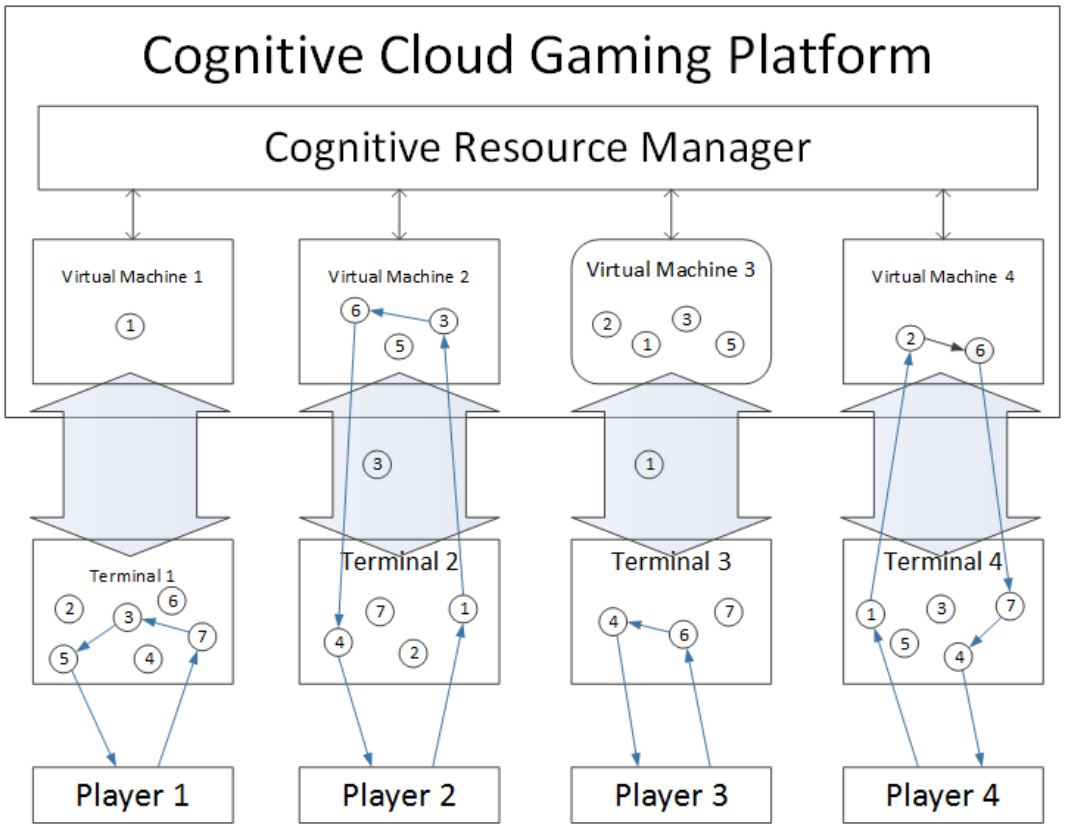


Figure 3.2: A Running Instance for Cognitive Cloud Gaming

In a typical scenario, the GaaS cloud hosts a number of terminals with a diversity of user-end devices and frequent changes in network QoS and cloud responses. The *Cognitive Resource Manager* monitors and assesses all terminals' working status and dynamically determines partitioning solution for each of them according to the system status, e.g., the available network bandwidth and remaining resources in the cloud, in order to provide cognitive capabilities across the cloud gaming system. As shown in Fig. 3.2, the four terminals host a different number of components locally, since their device status is distinct from each other. In the meantime, the *Cognitive Resource Manager*

schedules the Virtual Machine 2 and 3 to dispatch components to their corresponding terminals, since the available bandwidth allows the background downloads.

3.4 System Modeling

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

3.4.1 Game Components

In the cognitive cloud gaming platform, games consist of inter-dependent components that work collaboratively to provide gaming services for players. Similar to application's consumption graph in [78], Fig. 3.3 depicts an intuitive illustration that denotes the dependency of game components as a directed graph.

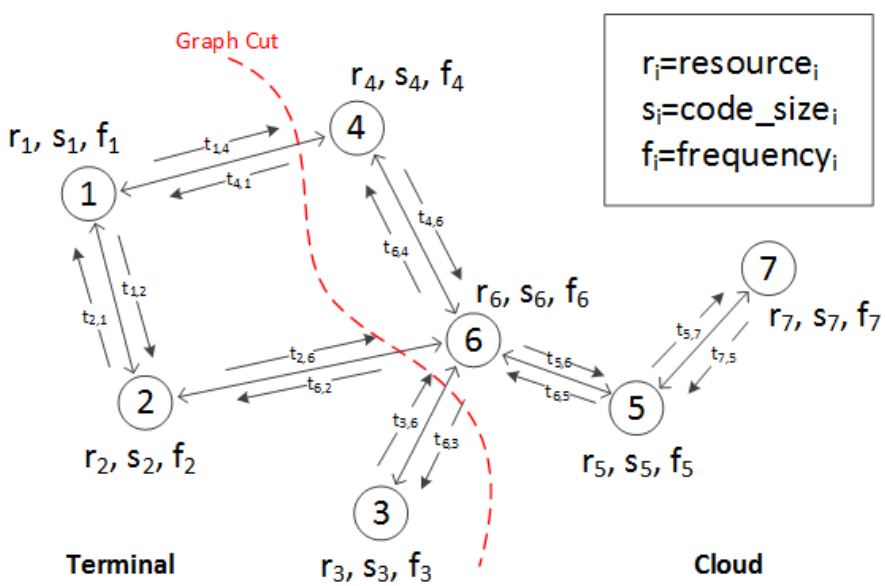


Figure 3.3: Components Partitioning for Proposed Cognitive Platform

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

between v_i and v_j. Each component v_i is characterized by the parameters listed in Table 3.1.

Table 3.1: 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
$f_{t_{i,j}}$	the frequency that v_i sends data output to v_j

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 depicted in Fig. 3.3, 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.

3.4.2 Formulation

To formulate the costs associated with partitioning of the game components, which properties are listed in Table 3.1, we further define specific parameters, including resource consumptions, code migration cost, and output transmission cost, as shown in Table 3.2. Note that the term "cost" in this chapter 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.

Table 3.2: 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

Note that t_i and c_i denote the component execution costs in the terminals and the cloud, which give an overall evaluation of resource consumptions r_i in CPU, memory and energy, etc., 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, etc., $\alpha_{i,j}$, $\beta_{i,j}$, $\lambda_{i,j}$ and $\theta_{i,j}$ denote message communication costs between components, subject to the amount of output data $o_{i,j}$, network QoS parameters, etc. Also, 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,

communications cost between all components C_c and code migration costs C_m as follows:

$$C_o = C_e + C_c + C_m \tag{3.1}$$

According to our definitions in Table 3.2, we derive:

$$C_e = \sum_{v_i \in \tau} t_i p_i + \sum_{v_i \in \sigma} c_i p_i \tag{3.2}$$

$$C_c = \sum_{\substack{v_i \in \sigma \\ v_j \in \sigma}} \alpha_{i,j} q_{i,j} + \sum_{\substack{v_i \in \tau \\ v_j \in \tau}} \beta_{i,j} q_{i,j} + \sum_{\substack{v_i \in \sigma \\ v_j \in \tau}} \lambda_{i,j} q_{i,j} + \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} \theta_{i,j} q_{i,j} \tag{3.3}$$

$$C_m = \sum_{v_i \in \tau} m_i \tag{3.4}$$

3.4.3 Optimal Partitioning Solution

The minimum overall cost C_o is subject to the various costs parameters denoted in Table 3.2. In this section, we derive the minimum cost for a special case, where the parameters for all v_i and e_i in Table 3.2 are all identical: we denote their constant values as $t, c, m, p, \alpha, \beta, \lambda, \theta, q$, 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 minimum spanning tree, 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.

Partitioning for Minimum Spanning tree

A minimum spanning tree (MST) or minimum weight spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree. In our context, it refers to a connected graph topology that contains $N-1$ edges, as illustrated by the example in Fig. 3.4.

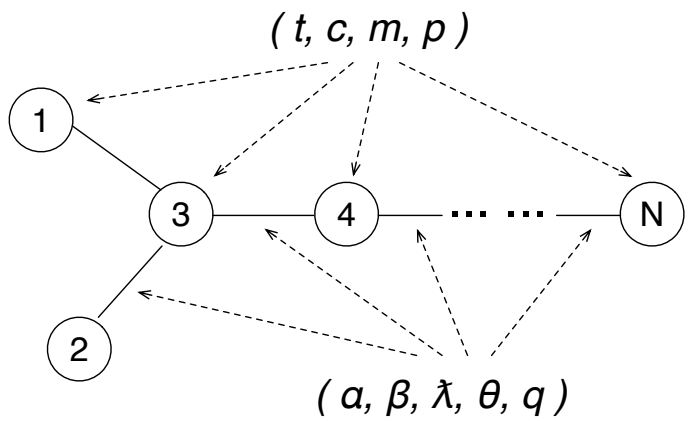


Figure 3.4: A Minimum Spanning Tree with N components

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:

$$C_e = x \cdot t \cdot p + (N-x) \cdot c \cdot p \tag{3.5}$$

$$C_m = m \cdot x \tag{3.6}$$

$$C_c = [(x-1) \cdot \beta + (N-x-1) \cdot \alpha + k \cdot (\lambda + \theta)] \cdot q \tag{3.7}$$

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 the following equation.

$$C'_o = [(t - c) \cdot p + m + (\beta - \alpha) \cdot q] \times 2 \tag{3.8}$$

Therefore, we see that if $C'_o \geq 0$, the value of x should be the minimum, i.e., 1, in order 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 to only 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:

$$m < (c - t) \cdot p + (\alpha - \beta) \cdot q \tag{3.9}$$

Partitioning for Complete Graph

A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge as illustrated in Fig. 3.5.

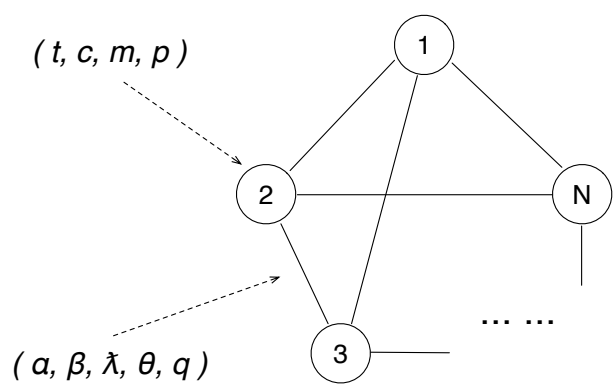


Figure 3.5: A Complete Graph with N components

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 (3.5) and (3.6), and C_c is formulated as follows.

$$\begin{aligned} C_c = [&\beta \cdot x(x-1)/2 + \alpha \cdot (N-x)(N-x-1)/2 \\ &+ (\lambda + \theta) \cdot (N-x)x] \cdot q \end{aligned} \tag{3.10}$$

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 follows:

$$\Delta = (N-2)[q \cdot (\beta - \alpha)(N-1)/2 + p \cdot (t-c) + m] \tag{3.11}$$

Given $N > 2$, we see that if $\Delta \geq 0$, in order 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:

$$m < (c-t) \cdot p + (\alpha - \beta) \cdot (N-1) \cdot q/2 \tag{3.12}$$

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 3 *Given a specific N, and 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 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, then we derive $\Delta C_c = C_c(G_B) - C_c(G_A)$ as follows:

$$\forall v_i, v_j \in L,$$

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

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 4 *Given a specific N, and 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 as 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 3*, we have $C_o^m(G_B) > C_o(B')$, where $C_o(B')$ is the overall cost of $G_{B'}$ with 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.

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:

$$C_o(G_M(N)) < C_o(G_R(N)) < C_o(G_C(N)) \tag{3.13}$$

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 illustrated in Fig. 3.6. It is mandatory that component A is hosted in the cloud and component B is executed in a terminal.

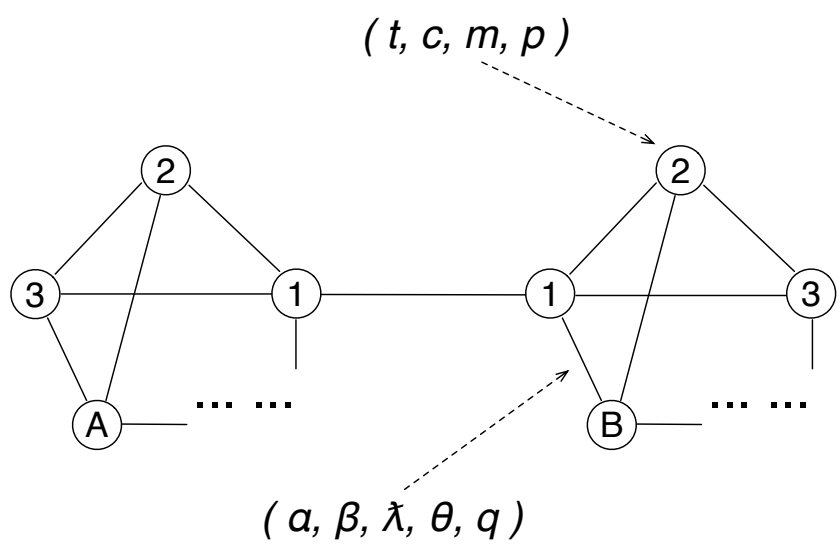


Figure 3.6: A General Graph with N components

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 (3.5) and (3.6), the minimized $\overline{C_c}$ is formulated as follows.

For $x_l < B$,

$$\begin{aligned}\overline{C_c(x_l)} =& \frac{A(A-1)\alpha q}{2} + \frac{(B-x)(B-x-1)\alpha q}{2} + \alpha q \\ &+ (\lambda+\theta)x(B-x)q + \frac{x(x-1)\beta q}{2}\end{aligned} \tag{3.14}$$

For $x_e = B$,

$$\overline{C_c(x_e)} = \frac{A(A-1)\alpha q}{2} + (\lambda+\theta)q + \frac{B(B-1)\beta q}{2} \tag{3.15}$$

For $x_g > B$,

$$\begin{aligned}\overline{C_c(x_g)} =& \frac{x(x-1)\alpha q}{2} + (\lambda+\theta)(N-x)(x-B)q \\ &+ \frac{(x-B)(x-B-1)}{2}\beta q + \frac{B(B-1)}{2}\beta q + \beta q\end{aligned} \tag{3.16}$$

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:

$$m(x_e - x_l) < [x_l(B - x_l) - 1](\lambda+\theta)q \tag{3.17}$$

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

$$\begin{aligned}\Delta_g =& [(t-c)p + m](x_e - x_g) \\ &+ [1 - (N - x_g)(x_g - B)](\lambda+\theta)q \\ &+ [\frac{A(A-1)}{2} - \frac{x_g(x_g-1)}{2}]\alpha q \\ &+ [\frac{(x_g-B)(x_g-B-1)}{2} - 1]\beta q\end{aligned} \tag{3.18}$$

Since $c \approx t$ and $\alpha, \beta \ll \lambda, \theta$, we can easily derive that $\Delta_g < 0$, 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.

3.5 Cloud Resource Management

As an assumption in previous sections, the execution and communication costs are considered 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.

3.5.1 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 3.3.

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

Table 3.3: 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

for terminal d is formulated as:

$$\mu(d) = \sum_{v_i \in \tau} t_i(d) p_i(d) + \sum_{\substack{v_i \in \tau \\ v_j \in \tau}} \beta_{i,j}(d) q_{i,j}(d) \tag{3.19}$$

Then the constraint on terminal processing resource is formulated as:

$$\forall d \in D, \mu(d) \leq P_T(d) \tag{3.20}$$

Terminal Network Constraint

The remote invokes of components introduce network cost, e.g., bandwidth, 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:

$$n_{i,j}(d) = \lambda_{j,i}(d) \cdot q_{j,i}(d) + \theta_{i,j}(d) \cdot q_{i,j} \tag{3.21}$$

and the constraint on terminal network resource is formulated as:

$$\forall d \in D, \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d) \leq N_T(d) \tag{3.22}$$

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:

$$\nu(d) = \sum_{v_i \in \sigma} v_i(d) p_i(d) + \sum_{\substack{v_i \in \sigma \\ v_j \in \sigma}} \alpha_{i,j}(d) q_{i,j}(d) \tag{3.23}$$

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

$$\sum_{d \in D} \nu(d) \leq P_C \tag{3.24}$$

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 follows:

$$\sum_{d \in D} (\sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d)) \leq N_C \tag{3.25}$$

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 work, the expected *processing latency* $l_p(d)$ is formulated as the sum of processing time which is proportional to the sum of processing resource consumptions in both terminal and cloud with efficient factor 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.

$$l_p(d) = f(\mu(d), \nu(d), F_T, F_C) \tag{3.26}$$

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 efficient factor F_N.

$$l_n(d) = \sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} g(n_{i,j}(d), T_{RTT}(d), F_N) \tag{3.27}$$

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

$$\forall d \in D, l_n(d) + l_p(d) \leq L_M(d) \tag{3.28}$$

3.5.2 Optimization Targets

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

and efficiency of the proposed system.

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 should be under the supervision of the QoS constraints, thus, to guarantee QoS satisfactory for all players.

$$\begin{aligned} &\text{Minimize:} && \sum_{d \in D} \nu(d) \\ &\text{Subject to:} && (3.20)(3.22)(3.24)(3.25)(3.28) \end{aligned}$$

Network Cost Minimization

The cloud gaming relies largely 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 cloud gaming service. Consequently, another optimization target *Network Cost Minimization* dynamically determines an optimized partitioning solution to minimize the terminals' average network cost.

$$\begin{aligned} &\text{Minimize:} && \sum_{d \in D} (\sum_{\substack{v_i \in \tau \\ v_j \in \sigma}} n_{i,j}(d) + \sum_{v_i \in \tau} m_i(d)) \\ &\text{Subject to:} && (3.20)(3.22)(3.24)(3.25)(3.28) \end{aligned}$$

Terminal Resource Cost Minimization

Terminals are considered 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, thus, provides lower energy consumption rate on players' devices.

$$\begin{aligned} &\text{Minimize:} && \sum_{d \in D} \mu(d) \\ &\text{Subject to:} && (3.20)(3.22)(3.24)(3.25)(3.28) \end{aligned}$$

Response Delay Minimization

The response delay to players' commands in gaming systems is defined as the time difference between the time when a player initiates a command and the time when the player receives the response from the game program. As one of most key factors that impact players' QoS, the response delay should be strictly controlled under a threshold (e.g., 200ms is "maximal tolerable" and 120ms is "hardly noticeable", as stated in [37]) to ensure acceptable gaming procedure. As indicated in [82], calculation of the response delay is completely different from conventional cloud gaming system. We provide *Response Delay Minimization* mode to minimize the terminals' average response delay with selected partitioning solutions.

$$\begin{aligned} &\text{Minimize:} && \sum_{d \in D} l_n(d) + l_p(d) \\ &\text{Subject to:} && (3.20)(3.22)(3.24)(3.25)(3.28) \end{aligned}$$

3.6 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 [83] approach, which have the potential to realize the advantages of the cognitive resource optimization proposed in this chapter in real-time scalable implementations.

The two proposed heuristic algorithms can be executed at either the run-time (on-the-fly) or pre-published stage (static approach) of a game. In contrast 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 pre-published stage, in order 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 states of the system (cloud, network and terminal) in real-time.

3.6.1 Local Greedy Approach

The intuitive motivation for considering a local greedy approach, which partitions the optimization problem into a set of sub-problems for all terminals, is its computational simplicity. After solving these sub-problems separately, the system concatenates all of the sub-solutions into a complete solution for the global problem. The computational complexity of the local greedy approach is significantly reduced to 2^N. Note that, since the

system needs to consider all QoS constraints, which include the bottleneck of the whole system, the concatenated solution might not fulfill the QoS requirements, thus, can not be considered global optimal. Besides, users are not independent of each other in the use of system resources, hence, the concatenated solution might not give the global optimal solution.

3.6.2 Genetic Algorithm-base Approach

In contrast, a Genetic Algorithm [83] (GA) is an adaptive heuristic search algorithm based on the evolutionary theory of genetics and natural selection, which simulates the survival of the fittest principle. This approach evolves a series of adaptive solutions, each described by a chromosome representing a particular genetic instance of the system, towards a desirable solution. The potential benefits of the GA approach include but not limited to: 1) controllable computational complexity: the system needs to response for situations in reasonable time, therefore, sub-optimal solutions with controllable decision time are required; 2) GA fits the distributed nature of cloud computing data centres: the chromosome operations can be processed in distributed manner and merged to a result, which is the essential idea of map-reduce algorithm; 3) GA produces a number of high-quality chromosomes according to the selection criterion. Therefore, these chromosomes are reusable when new the environment changed, which tends to speed up the convergence.

Encoding: In a GA system, each solution to the problem is described as a chromosome, an individual with a particular genetic instance in the nature. In this work, we devise a chromosome consisting of $N \times C$ bits: "1" represents the corresponding component is hosted in the cloud while "0" represents the component is executed in the terminals. Fig. 3.7 shows an illustrative example for the substantiation of solution encoding. With the

chromosome, the system places a component for execution, e.g., the Yth component of terminal X, by looking up the bit $C * (X - 1) + Y$ in the chromosome.

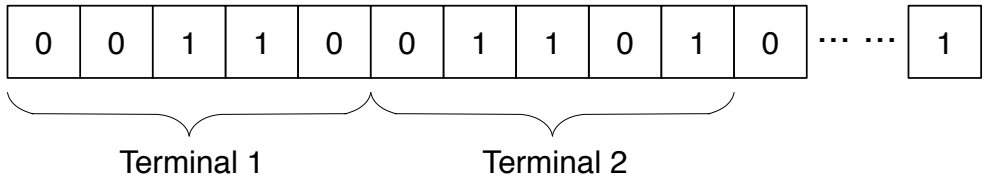


Figure 3.7: Chromosome Encoding Method for Cloud Resource Management

Cloning: The cloning operation is to simulate the growth of chromosome based on the initial population. In our algorithm, initial population K is created by random procedures for the purpose of gene diversity, while the cloning is a copying process that produces redundant chromosome, according to the *expending factor* F_e. In general, we set $F_e = 2$, implying the chromosome quantity is doubled in one cloning operation.

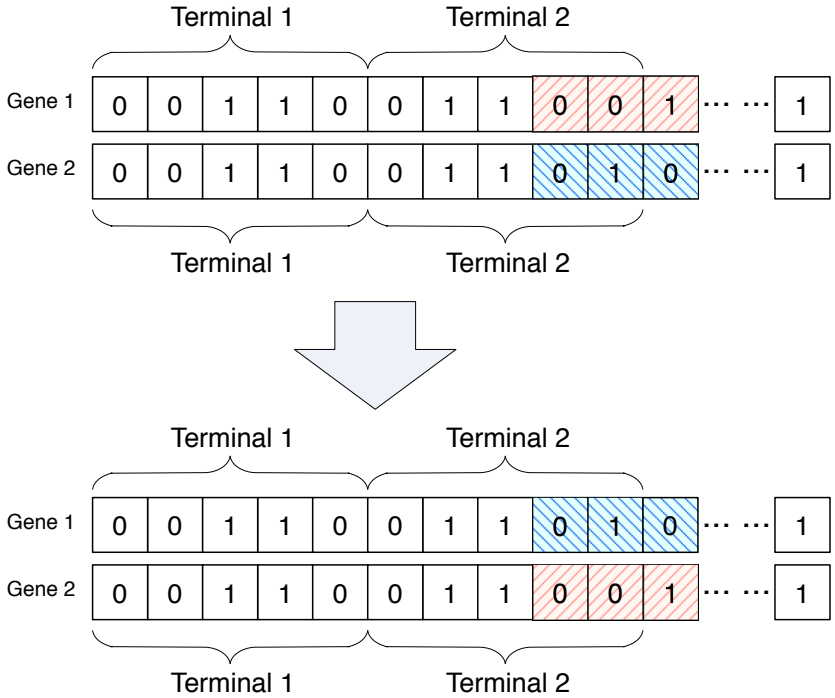


Figure 3.8: Chromosome Crossover for Cloud Resource Management

Crossover: A crossover operator is a key component in GA. It imitates the way of

natural biological evolution. There are several crossover schemes have been proposed, such as one-point crossover[84] and multi-point crossover[85]. In our approach, the crossover is applied to two random chromosomes G_1, G_2 with a probability of $P_{crossover}$, as shown in Fig. 3.8. We randomly select two integers $r_1, r_2 \in [0, N \times C)$ to represent the starting and ending bit of the crossover segment and switch the corresponding segments in G_1 and G_2.

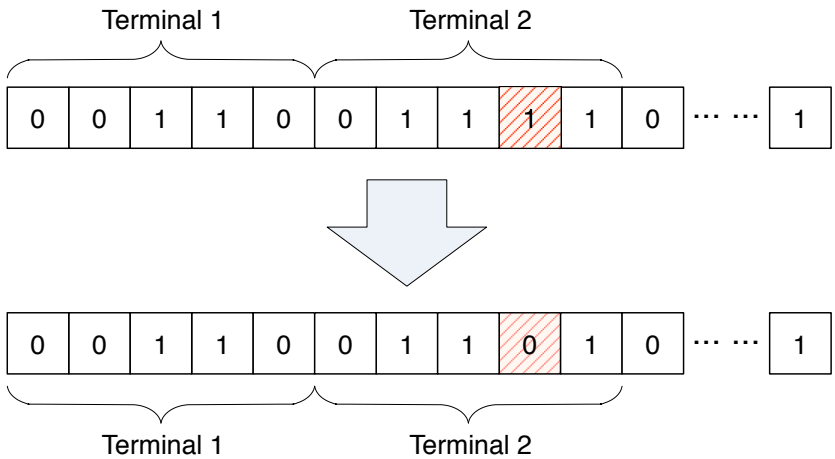


Figure 3.9: Chromosome Mutation for Cloud Resource Management

Mutation: The mutation operator is used to accommodate the variety of the genes so that the discovery of new (hopefully better) solutions is possible. In our approach, mutation is performed on duplicated chromosomes with a probability of $P_{mutation}$. The operator randomly selects an element in the chromosomes and switches its value to the opposite: "1" to "0" or "0" to "1", as illustrated in Fig. 3.9.

Correction: After crossover and mutation operations, some chromosomes might not be valid anymore, according to the system setting. For instance, some components are mandatory to be executed in the terminals, e.g., the input processing module, while some others are forced to be implemented in the cloud, e.g., the networking module. Therefore, a correction operator is required to ensure that all chromosomes are valid. In this work, a check process in special bits will be performed as the correction operator.

Selection: The chromosome selection is to simulate the survival of "stronger"

chromosomes from the nature selection. Those chromosomes with better genes, according to the selection criterion, will be the parents of next generation of populations. In this work, we rank all chromosomes with the result value of fitness function and retain top genes of initial K population for reproducing the offspring generation.

Fitness Function: A fitness function is the selecting criteria that representing a specific optimization target for the chromosomes. As a cognitive system, the fitness function is always dynamic and adaptable to the overall system performance.

3.7 Simulations

3.7.1 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 function f and g described in Section 3.5.1 follows the formula below:

$$f(\mu(d), \nu(d), F_T, F_C) = \mu(d) \cdot F_T + \nu(d) \cdot F_C \tag{3.29}$$

$$g(n_{i,j}(d), T_{RTT}(d), F_N) = n_{i,j}(d) \cdot T_{RTT}(d) \cdot F_N \tag{3.30}$$

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 3.4. Note that, *Communication Probability* is defined as the probability of a communication occurs between two components. Higher valued *Communication Probability* between two components indicates a higher possibility that the two components will invoke

each other. This is determined by the functionality of the components and their execution sequences. In contrast, *Data Transmission Probability* denotes the transmission probability between two components, given the invocation between these two components existed. It indicates the how often one component will invoke the other during the game execution. It is determined by the functionality of the components and also the players' behaviors in playing the games. We can also notice from the table that, the communication cost from the terminal to the cloud is a bit larger than the one from the cloud to the terminal, since we assume that the terminal needs to cost more energy to initiate a data transmission, especially when they are powered by batteries.

Table 3.4: Parameters for 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 Probability	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

The default simulation parameters of terminal devices are listed in Table 3.4, which represents the wide range of terminals, from stationary computers to mobile devices connected to wireless networks.

All random variables listed in above tables follow the uniform distribution. To simplify the QoS measurement, we set 120 milliseconds(ms) as the maximal tolerable latency value as the indicator of QoS. All simulations are repeated for 2000 times with distinct random seeds to yield an average value.

Table 3.5: Parameters for 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

3.7.2 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.

Overall Cost Boundary

In this section, we study the overall cost of a specific game in different partitioning solutions. As depicted in Fig. 3.10, 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 a distinct number of components. As we can see from the figure, the gap between maximum and minimum feasible values is becoming smaller when the number of components increases.

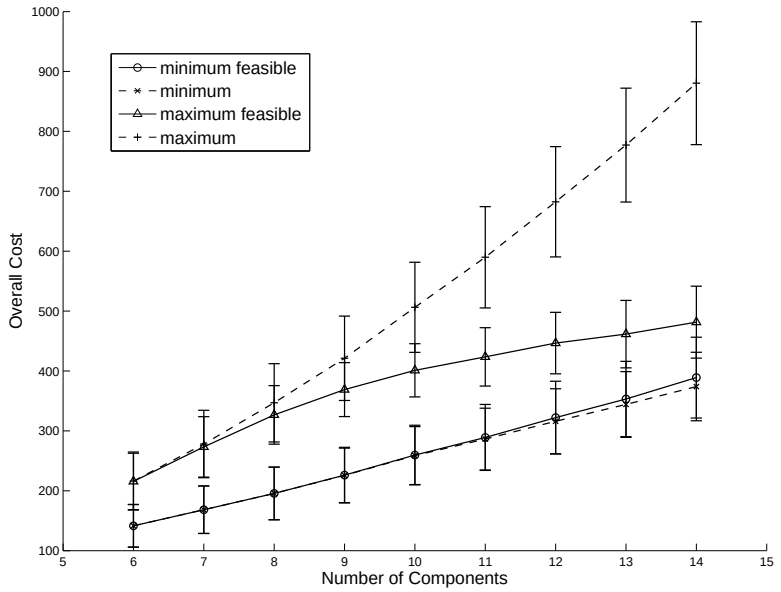


Figure 3.10: Effect of Number of Components on Overall Cost

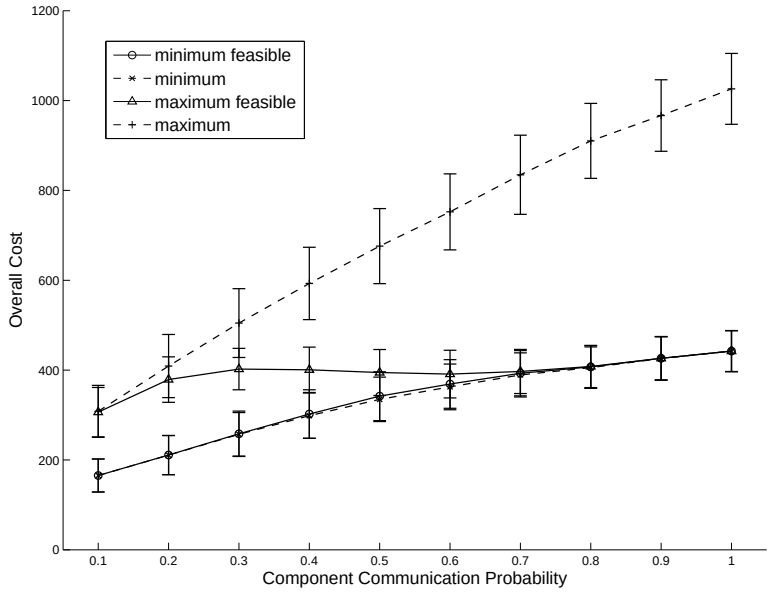


Figure 3.11: Effect of Component Communication Probability on Overall Cost

The similar trend is also discovered in Fig. 3.11, which illustrates the effect of component communication probability on overall cost. Apparently, increased

communication probability indicates the increases in network costs in optimal partitioning, which exclude more infeasible schemes with higher overall cost from the results. We also plot the error bar for these two figures, with the confidential interval of 95%.

Effect on Terminal Incapable Rate

In our simulation, the games' component structures are randomly created. Hence, some of them might not have a feasible partitioning scheme to fulfill the QoS requirements. In this chapter, 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 *Pure-Cloud*, the conventional partitioning solutions for cloud gaming, to our proposed cognitive solutions over the *Terminal Incapable Rate*.

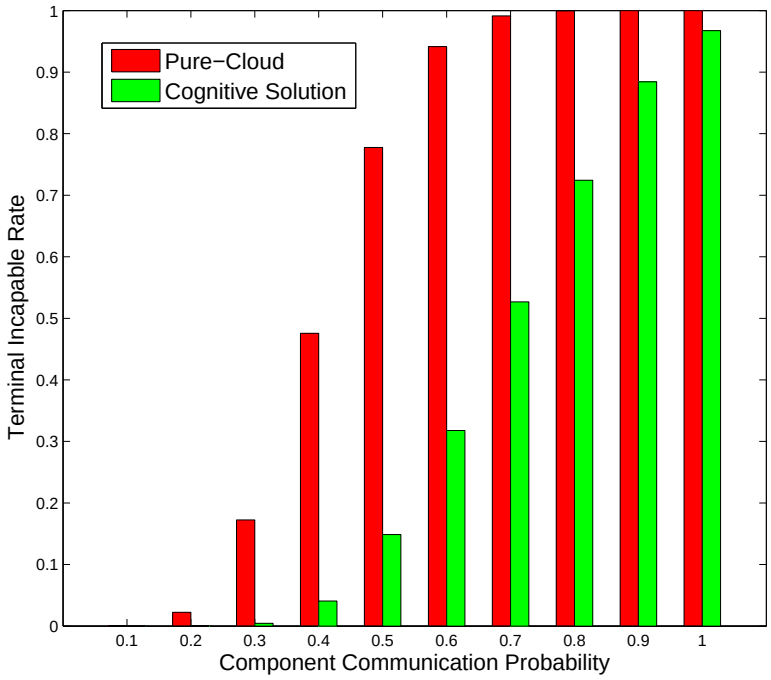


Figure 3.12: Comparison on Terminal Incapable Rate

As shown in Fig. 3.12, as the communication probability arises, more network communications cause higher overall response delay. For the component communication probability of 0.5 and 0.6, The *Terminal Incapable Rate* for *Terminal Incapable Rate* 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.

Effect of Communication Probability

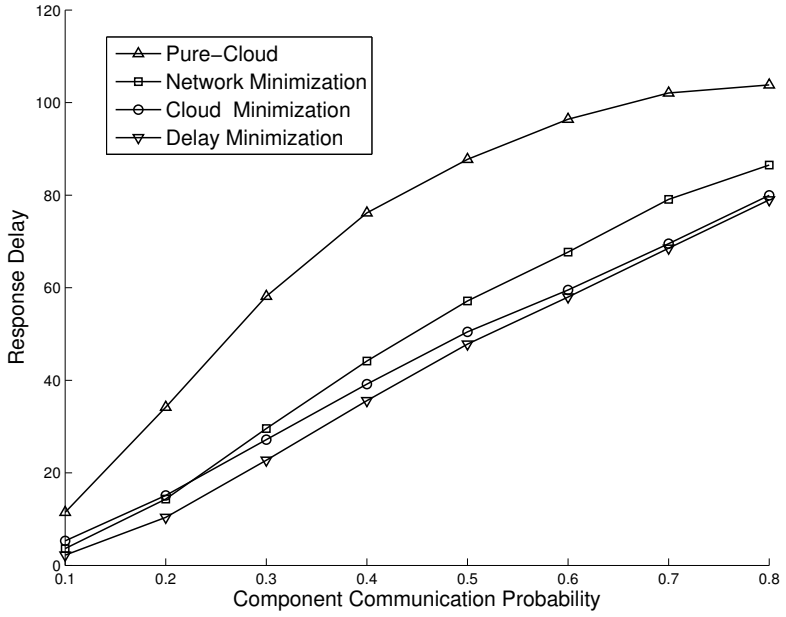


Figure 3.13: Effect of Communication Probability on Response Delay

Fig. 3.13 illustrates the impact of communication probability on decomposed games to the average response delay. Apparently, the average response delay becomes larger when the communication probability increases. In order to demonstrate the different optimization targets, we optimize the system from the aspects of terminal, network, cloud and delay. In fact, the terminal minimization is the Pure-Cloud mode. As we can see,

Delay optimization achieves the best performance comparing to other schemes.

Effect of Component Degree Variance

However, other than communication probability, the density and skewness of the communication links between components are also very important factors to impact the game performance. In order to describe the communication density and skewness, we calculate all communication degrees for each component and derive their communication variance as an evaluation criterion. As depicted by Fig. 3.14, we derive the average value of the response delays for all four optimization schemes at different variance intervals. In our experimental settings, the four optimization schemes reach their peaks at minimum variance interval of 0 to 2, while they all decreased to the bottom value at the maximal variance of 10 to 12. This simulation result implies that the more imbalance the communication network is, the lower responsive delay our cognitive system achieve.

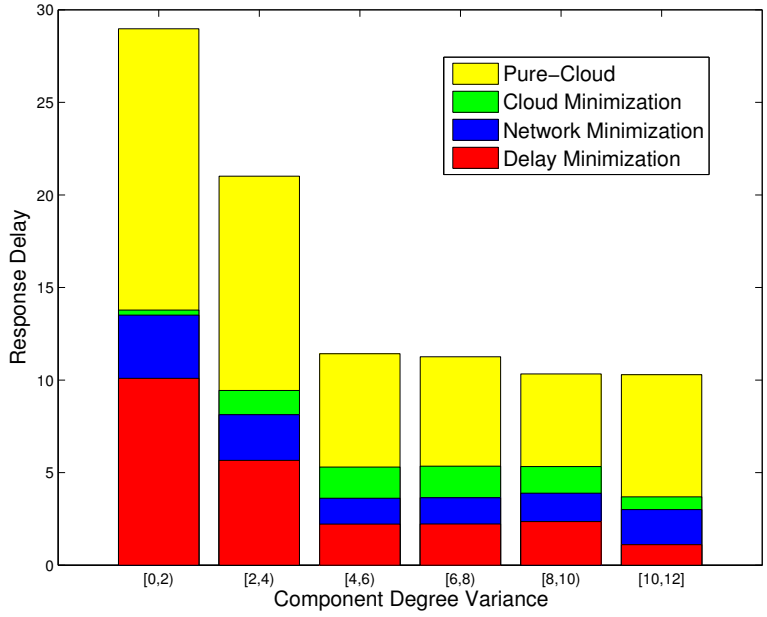


Figure 3.14: Effect of Communication Variance on Response Delay

Tradeoff study: cloud and terminals

Here we study the correlations among cloud and terminal cost in a particular partitioning solution. In contrast to optimized solution selection, we listed all possible solutions and calculated their cloud, network and terminal costs to illustrate their relationships. We studied that the cloud cost is inversely proportional to terminal cost, as shown by the linear fitting in Fig. 3.15.

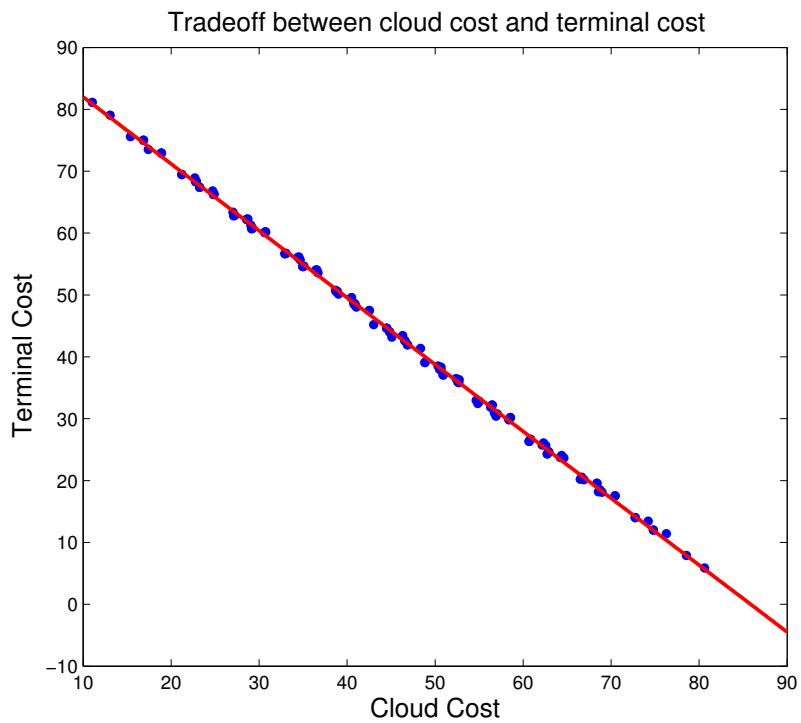


Figure 3.15: Tradeoff between cloud cost and terminal cost

3.7.3 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 3.6 below.

Table 3.6: Parameters for Genetic Algorithms 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

GA Convergence

With 60 devices, we conduct an experiment on minimizing the response delay with the proposed GA solution. As shown in Fig. 3.16, the system reaches the theoretical boundary of minimal response delay after 450 offspring generations. This is an evidence of the convergence of the proposed GA solution.

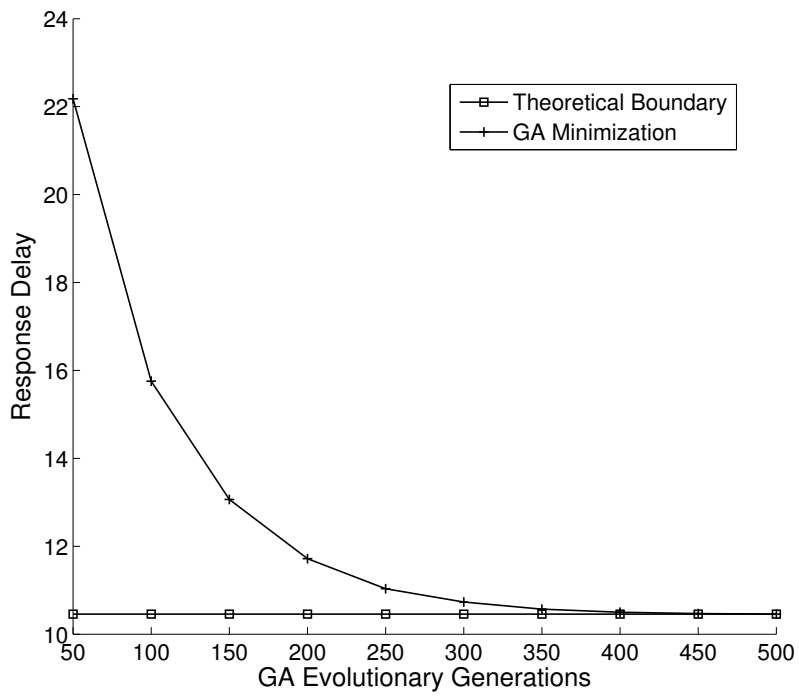


Figure 3.16: GA convergence on evolutionary generations

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 cost achieved by the local greedy algorithm and GA with various generation iterations of 100, 200, and 300, respectively. For comparison, we also illustrate the average network cost of *Pure-Cloud* solution, and Optimal, the systemic minimal average network cost. Note that, in order to eliminate 2^{CN} search for optimal solutions, we set up an infinite loop for GA iteration. If an additional 2000 iterations yield the same result, we consider 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 @2.70GHz CPU and 4.0 GB Internal Memory (RAM). According to our experimental settings, even though for 100 devices, local greedy and GA with 100 and 200 iterations can be completed within 1 second, while GA of 300 iterations can be done in 2 seconds. In contrast, the 2000 iterations of GA consume around 28 seconds with 100 devices, while only 1 second 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 further shortened.

As shown in Fig. 3.17, with large scale simulations, the local greedy algorithm provides around 19.5% to 10.5% decrease in average network cost compares to the *Pure-Cloud* solution, while the series of GA solutions demonstrates even higher efficiency in optimizing the average network cost. It can be observed that when the device quantity is relatively small, e.g., 10, a small number of generation iterations (e.g., 100) is able to achieve the 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.

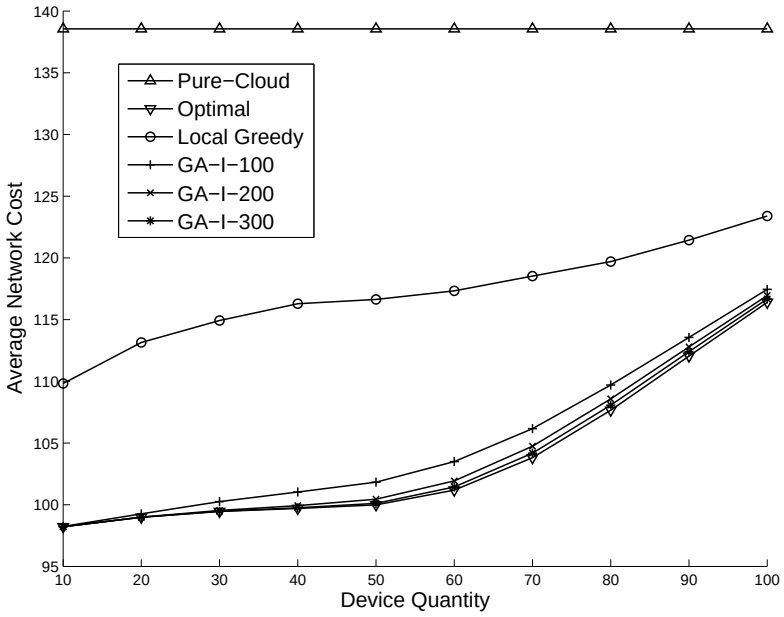


Figure 3.17: Performance Evaluation on Network Cost Minimization

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 partitionings 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.

Cloud Resource Cost Minimization

We also perform simulation on the optimization target of cloud resource cost minimization to compare the efficiency of *Pure-Cloud* solutions, local greedy algorithm and series of GA schemes.

Fig. 3.18 shows the comparison on the average throughput for the terminal devices. Apparently, for *Pure-Cloud* mode, the cloud cost is proportional to the device quantity.

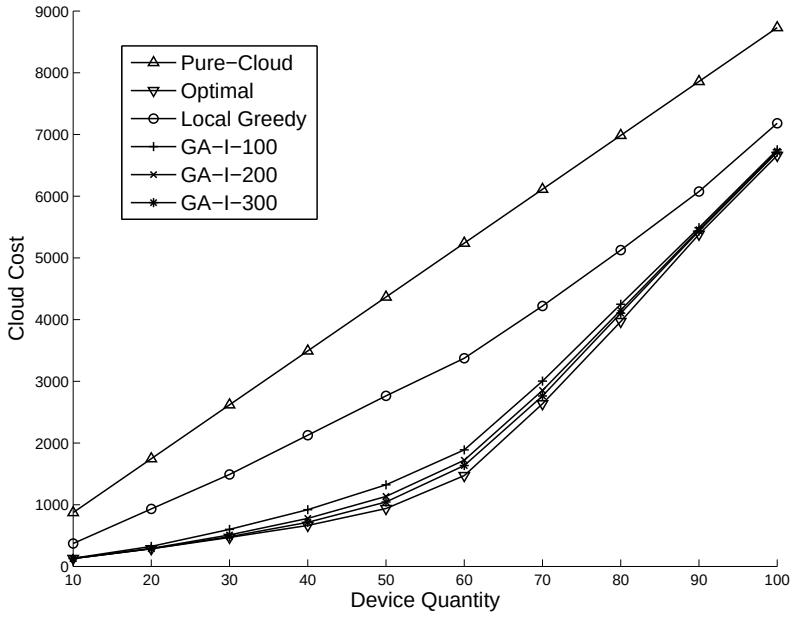


Figure 3.18: Performance Evaluation on Cloud Resource Cost Minimization

In order to support 100 terminals, the gaming server needs 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 around 6800 units are required, which is only 75.5% of conventional *Pure-Cloud* 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 12000 cloud network resource, the resource optimization achieves the best performance when device quantity is 60, where the Local Greedy algorithm reduce the cloud cost from 5300 units to 2800 units and the GA solution only consumes 1500 units, representing declines of 47% and 71.7%, respectively. Similar to the previous simulation, for the purpose of illustrating the variance, we depict

the box plot for selected local greedy scheme in Fig. 3.19.

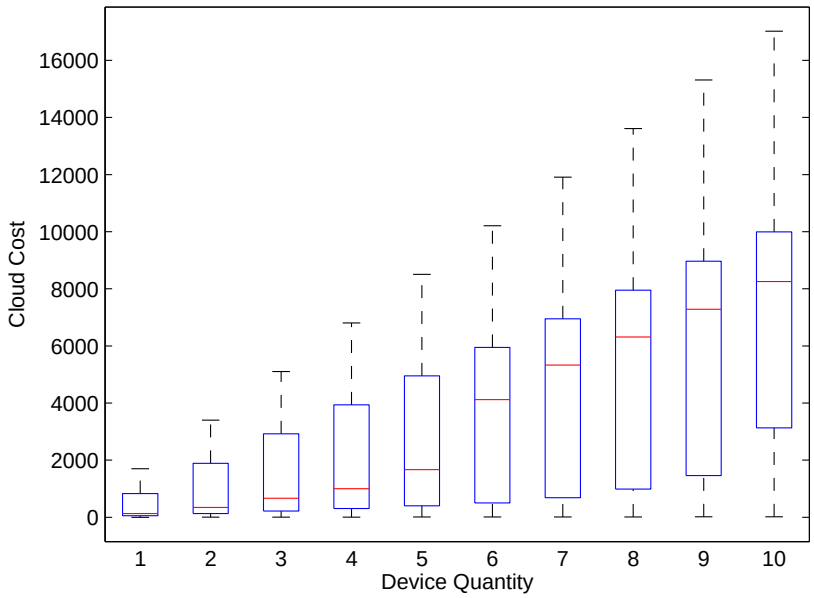


Figure 3.19: Variance of Local Greedy Scheme Simulation Results

3.8 Summary

The decomposed cloud gaming platform introduces a flexible component allocation solution to optimize cloud gaming service provision. In this chapter, we have presented system modeling and simulation 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. The studies in this work is limited to the lack of explicit modeling for component characteristics in game program, which is still blank in this area. Therefore, future research in this topic should be led by trace measurements of real systems.

Chapter 4

Decomposed Ubiquitous Cloud Gaming System

4.1 Introduction

As discussed in Chapter 3, decomposition provides a potential solution for cognitive cloud gaming. 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, hence meeting the challenges caused by the limitations of the mobile networks. However, how to design such a decomposed software system for games is still an unanswered question, given a number of critical issues in system implementations. In this chapter, we discuss all of these detailed issues from an engineering perspective and develop testbed and game prototypes to validate our theoretical proposal in Chapter 3. The outline of this chapter is as follows. We discuss the design objectives and principles of the cognitive platform for ubiquitous cloud games in Section 4.2 and 4.3, respectively. The design and implementation of the cognitive platform are described in Section 4.4 and 4.5, respectively. Experiments on optimizing the overall latency and enabling partial offline execution are conducted in Section 4.6. Section 4.7 summarizes the chapter.

4.2 System Objectives

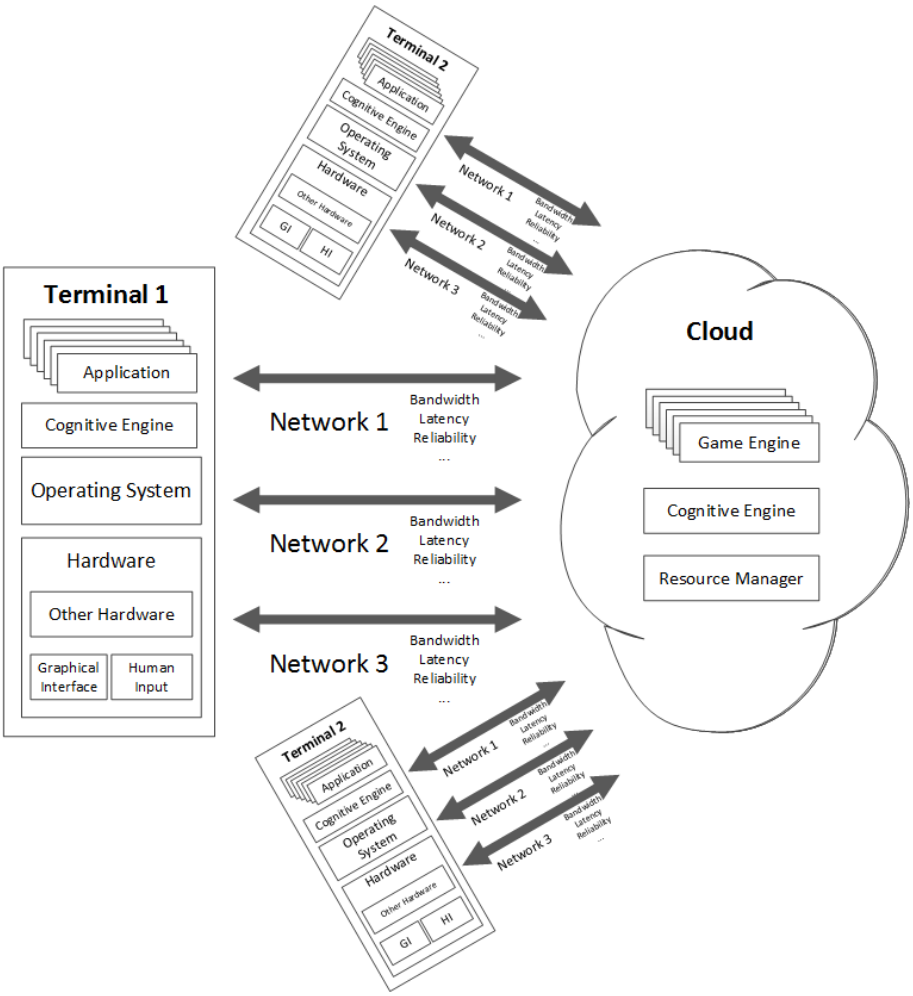


Figure 4.1: Architecture Framework for Cognitive Mobile Cloud Gaming

Cognitive systems [86] are attractive for the heterogeneous environment we are considering. The situation-aware architecture of cognitive system monitors and assesses the working environment to predict and make decisions for the providing services, and refine future decisions by learning from the achieved results. To the best of our knowledge, a QoE-oriented cognitive system is not available for cloud gaming systems. To provide cognitive capabilities across the cloud gaming system, we need to overcome the diversity of user-end devices and frequent changes in network QoS and cloud

responses. We use the concept of cognitive system design (i.e., act, learn, adapt) to realize our proposed situation-aware cloud gaming platform. Our objective is to develop an architectural framework that is cognitive of resources and characteristics of the cloud, the access network, the end-user devices and the players' behaviors, to enable dynamic utilization of these resources.

As shown in Fig. 4.1, we envision a cognitive platform with a novel capability to learn about the game players' environment (i.e., 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 4.1.

4.3 Design Principles

4.3.1 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. There are different types of "Decomposition" defined in computer sciences. In this chapter, the term of decomposition is defined as "breaking a large system down into progressively smaller classes or objects that are responsible for some part of the problem domain". In fact, algorithmic decomposition is a necessary part of the object-oriented analysis and design [87]. Due to the distributed nature of cloud computing resource infrastructure, decomposition of an application program provides the potential improvement on intra-cloud execution efficiency. In this section, we discuss two level of decomposition for our proposed system.

Table 4.1: High-Level System Requirement

Requirement	Definition	Responsive
SR_1 Click-and-Play	To enable immediately game play when the players select their favourite game from platform portal	Onloading Manager
SR_2 Performance Evaluation	To measure, evaluate and predict the overall system performance, including CPU load, memory usage, link bandwidth utilization, 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 Adaption	To intelligently adapt its gaming services to system performance, such as changing network conditions (QoS) and different players' distinct behaviors, to keep user QoE above a prescribed threshold.	Cognitive Engine
SR_5 Partial Offline Execution	To lessen the dependency on network connection by facilitating partial offline gaming in specific scenes.	Message Redirector

Fine-Grained Decomposition

Fine-grained decomposition refers the design patterns that segment the whole game program into tiny components, i.e. function/methods, as depicted by an example in Fig. 4.2.

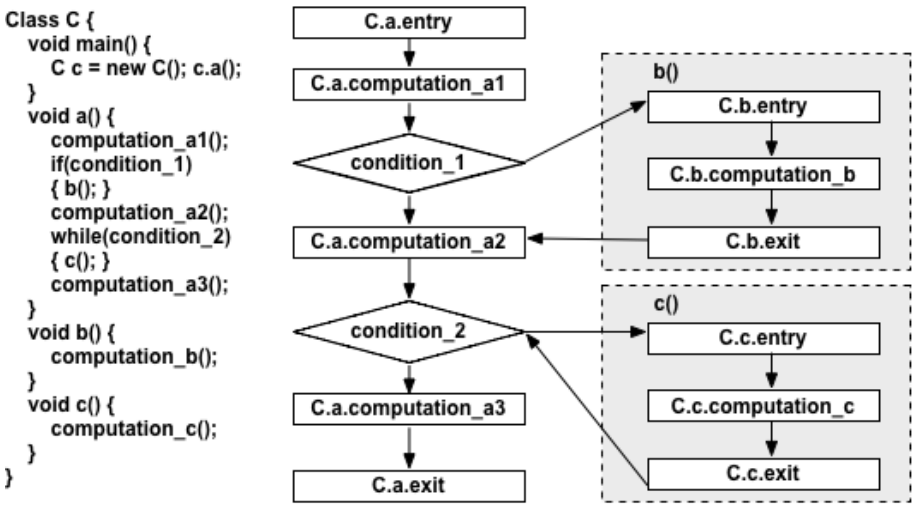


Figure 4.2: Fine-Grained Decomposition Example

In this program, method $b()$ and $c()$ are invoked by the function $a()$ with *if* and *while* conditions, respectively. Given method $a()$ is executed in a resource-restricted mobile terminal, hosting $b()$ and $c()$ in the cloud might be beneficial, if the computational cost of $b()$ and $c()$ are relatively higher than their overall communication costs with $a()$. Fine-grained decomposition provides a huge quantity of tiny components, therefore, it exhibits most flexible partitioning schemes, which leads to more opportunities in seeking optimal component allocation solution.

Nevertheless, the fine-grained decomposition model is also limited in some respects. The most critical issue in method-level decomposition is the state migration problem. Conceptually, if a component remotely invokes another one, i.e. a terminal hosted component calls a method executing in the cloud, the component need to collect native context for transfer as well. However, the complexity of serializing this information for

network transfer and also the complexity of parsing these data after transmission to the destination are significantly higher, given processor architecture differences, differences in file descriptors, etc. As a result, the CloneCloud system proposed in [80], which adopts the fine-grained decomposition modality, only supports migrating at execution points where no native state (in the stack or the heap) need be collected and migrated.

Coarse-Grained Decomposition

In contrast to fine-grained decomposition, a coarse-grained decomposition partitions the game program into a number of functional independent and stateless components. These components are often composed of a set of objects and methods, which work collaboratively within the scope of the component. See Fig. 4.3 for an illustration.

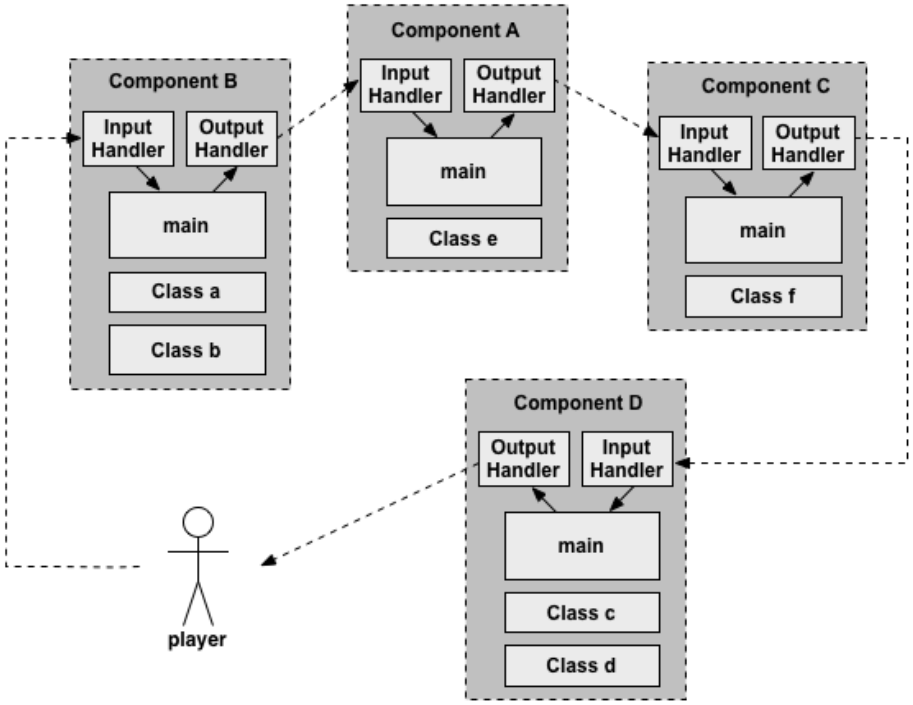


Figure 4.3: Coarse-Grained Decomposition Example

In this program, the instance of *Component B* is activated by a control instruction from the player. After processing with its designated class and methods, it calls its

consequent component, which is *Component A* in this context, for further manipulation. After successive processing by *Component C* after *Component D*, the player received the resulting gaming content. Since these components are in charge of relatively independent functionalities, their native states are always invisible to each other, which eliminates the state transfer problem in fine-grained decomposition.

However, the coarse-grained decomposition still has open issues in game development. In comparison to fine-grained design, the coarse-grained decomposition results in fewer components, which might affect the efficiency of cognitive resource allocation. In addition, it is relatively difficult for game developers to write the game program, since these components are strong coupling to each others. This requires the software architects to abstract the main building blocks of a specific game in prior and to pre-define all unified interfaces between the components.

4.3.2 System Workflow

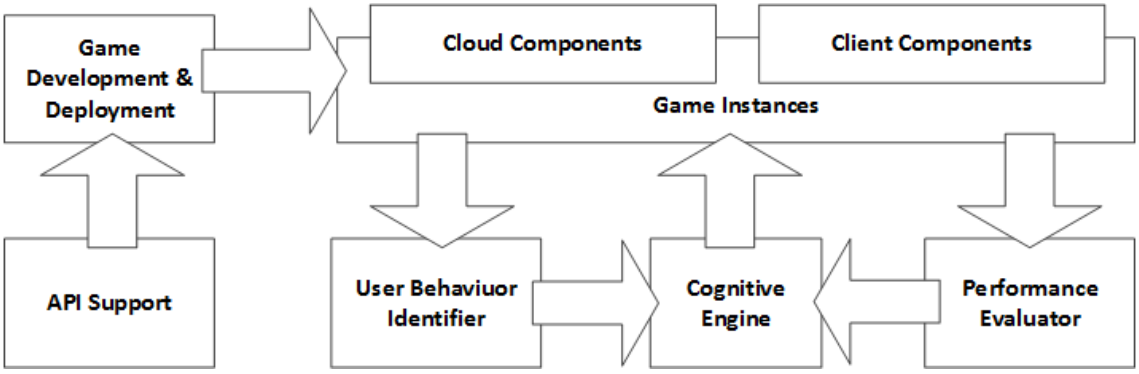


Figure 4.4: Work Flow of Decomposed Ubiquitous Cloud Gaming Platform

Fig. 4.4 illustrates a workflow of proposed cognitive ubiquitous cloud gaming platform. In the first stage, the Cognitive Ubiquitous Cloud Gaming platform should provide API support for game developers. With APIs, the game developers write and deploy the source codes before the game instances are launched for the connection

requests from the players. During the gaming sessions, the Performance Evaluator focuses on collecting the execution efficiency parameters, such as execution latency, network round-trip time, etc. Meanwhile, the User Behavior Identifier learns players' interactions with components by performing statistical analysis on inter-components invocations. The results of performance evaluation and players' user behavior are sent to the Cognitive Engine as the references of dynamic partitioning. In other words, the cognitive engine will dynamically assign a set of components to be executed in the cloud, while migrates the others to the terminals, in order to maximize the overall system performance.

4.3.3 Dynamic Partitioning

To help readers of this chapter understand the dynamic partitioning of UBCG Testbed, we illustrate a sequence diagram for the cognitive engine of UBCG Testbed in Fig. 4.5. To start the gaming, the cloud server first dispatches a JavaScript engine (including a small portion of game instance) to the terminal, while launching the game instance in the cloud. During the gaming session, the gaming instance in terminal keep sending status statistics to the *Cognitive Engine*, which analyze the system performance and acknowledge the *Partitioning Coordinator* its decision of partitioning. This decision will instruct the *Partitioning Coordinator* to redirect all control messages and inter-component messages, in order to facilitate dynamic partitioning. In other words, this is the key mechanism that enables the proposed environment-aware adaption. Note that, there is an optional *onloading* process when *Partitioning Coordinator* receives decisions from *Cognitive Engine*. This process works if the terminals are a lack of sufficient components to perform optimal partitioning.

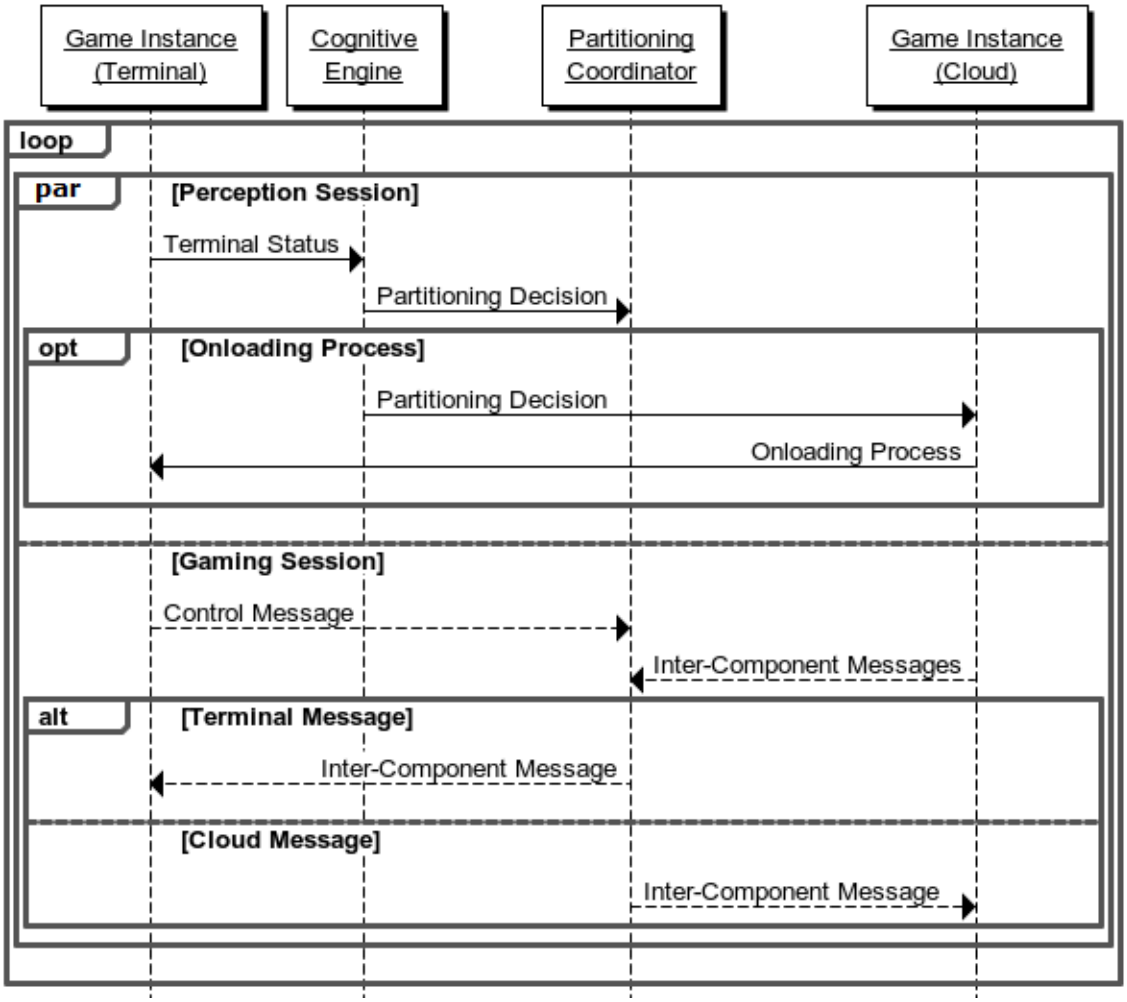


Figure 4.5: Sequence Diagram for Cognitive Engine

4.3.4 Response Delay

The game program is a latency-sensitive interactive system. The response delay to players' commands is defined as the time difference between the time when player initiate a command and the time when player receive the response from the game program. In general, the maximal tolerate response delay $d_{tolerate}$ is 120 ms. Therefore, to guarantee the response delay is a critical issue in cloud gaming system.

The work [6] segments response delay of a conventional RR-GaaS cloud gaming system into three components: network delay, processing delay and playout delay. However, to

calculate the response delay in a decomposed cloud game is completely different.

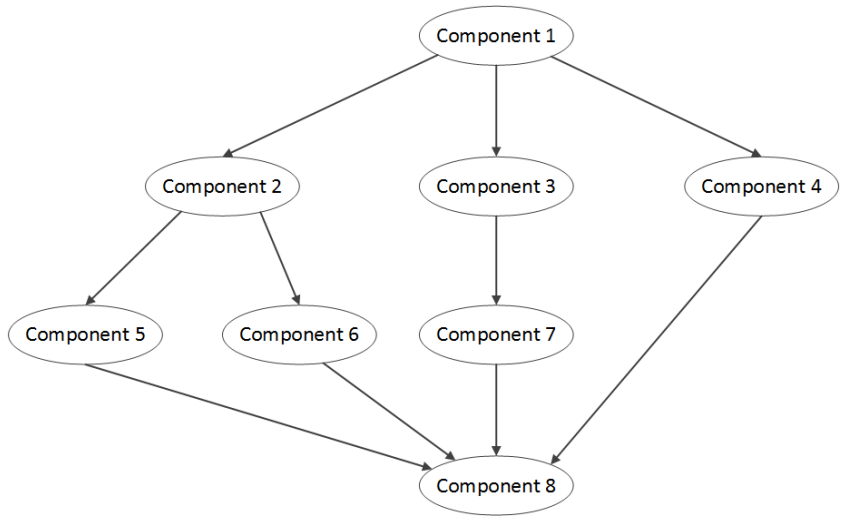


Figure 4.6: A Component Communication Tree

Take the communication tree in Fig. 4.6 as an example, where the edges' weight refers to their communication costs. Denote *Component 1* and *Component 8* as the input component and output component in a decomposed gaming system, we hereby define a message path through *Component 1* to *Component 8* as a *Response Cycle*. Accordingly, there are 4 response cycles in Fig. 4.6. Therefore, we formulate the response delay d_c for response cycle c as:

$$d_c = \sum_{n \in c} e_n + \sum_{i \in c, j \in c} c_{i,j} \tag{4.1}$$

where $c \in C$ is the set of response cycle in the game program, e_n is the execution latency for component n and $c_{i,j}$ is the communication latency between component i and component j. However, the response delay can be various, according to the content of the command, the parameters of the gaming scenes and the reacting logics defined in the components. Therefore, to satisfy the restrictions of maximal tolerate response delay, the system need to locate the response cycle with maximal response delay d_{max} by traversing

all response cycle in C:

$$d_{max} = \max_{c \in C} d_c \tag{4.2}$$

Hereby we denote the constraints of response delay on a decomposed cloud game as:

$$d_{max} \leq d_{tolerate} \tag{4.3}$$

4.3.5 Partial Offline Execution

One of the most critical problems for mobile cloud gaming is the conflict between strong network dependency and unstable network connectivity. In all existing cloud gaming frameworks, the gaming session will be suspended or even destroyed, once the mobile device loses its network connection to the cloud. However, players are not interacting with each other all the time during the gaming session, even in those multi-player games. In a typical Massively Multiplayer Online Role-Playing (MMORPG) game, the avatar spends a remarkable amount of time in monster hunting by him/her-self, for the sake of level-up and outfit gathering [88]. The players will never be happy if they lose valuable items in the battlefield due to the network access problem. Our proposed platform focus on the solution to these scenarios.

Fig. 4.7 demonstrates a case of redirection service provided by the proposed platform: the mobile-executed component 7 is trying to activate component 3 in the cloud with an output message, while the network connection to the cloud is temporarily lost. Rather than suspending the gaming session, the terminal locates component 3 on the mobile device and redirects the output message to this local copy. Thus, the player will not be disturbed by the temporary disconnection of Internet. This approach is called "Partial Offline Execution". Note that once the mobile device recovers its network access, a *Synchronization Controller*,

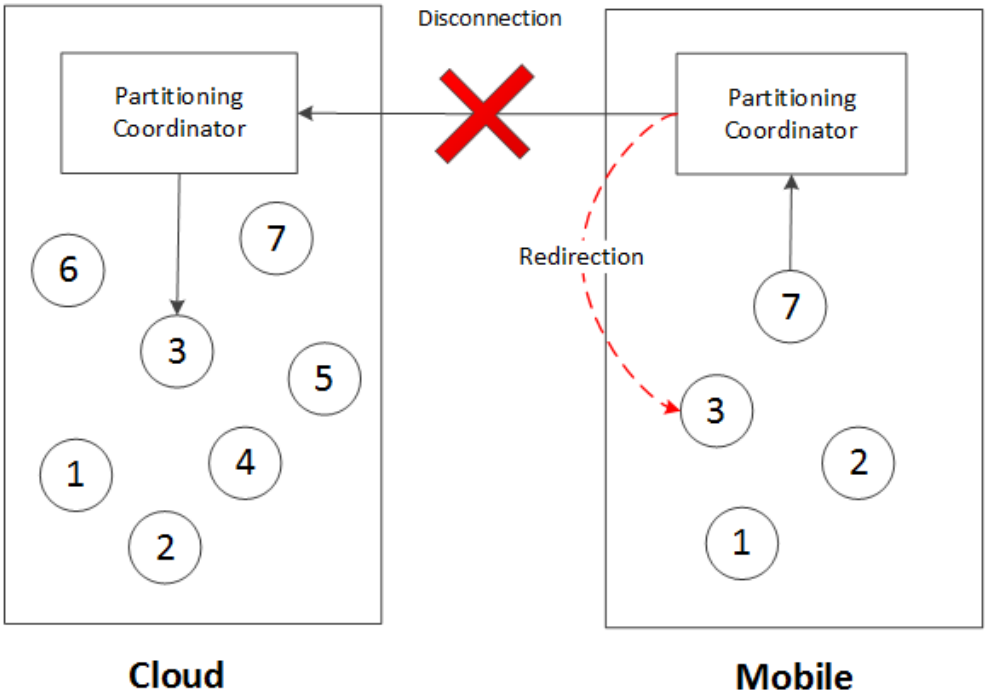


Figure 4.7: Redirection Service in Disconnection to the Cloud

which is aware of the data modifications, should perform data synchronization with the cloud server.

4.4 Framework Design

Given the above requirements, we define the main building blocks of the architectural framework on both the cloud-side and terminal-side, and identify the prevalent standards that are applicable to the interfaces between these building blocks.

Fig. 4.8 illustrates the elements of the proposed cognitive platform that facilitate the dynamic partitioning. *Execution Monitor*s implemented on both the 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, execution time, are reported to the *Cognitive Decision Engine*, where cognitive resource

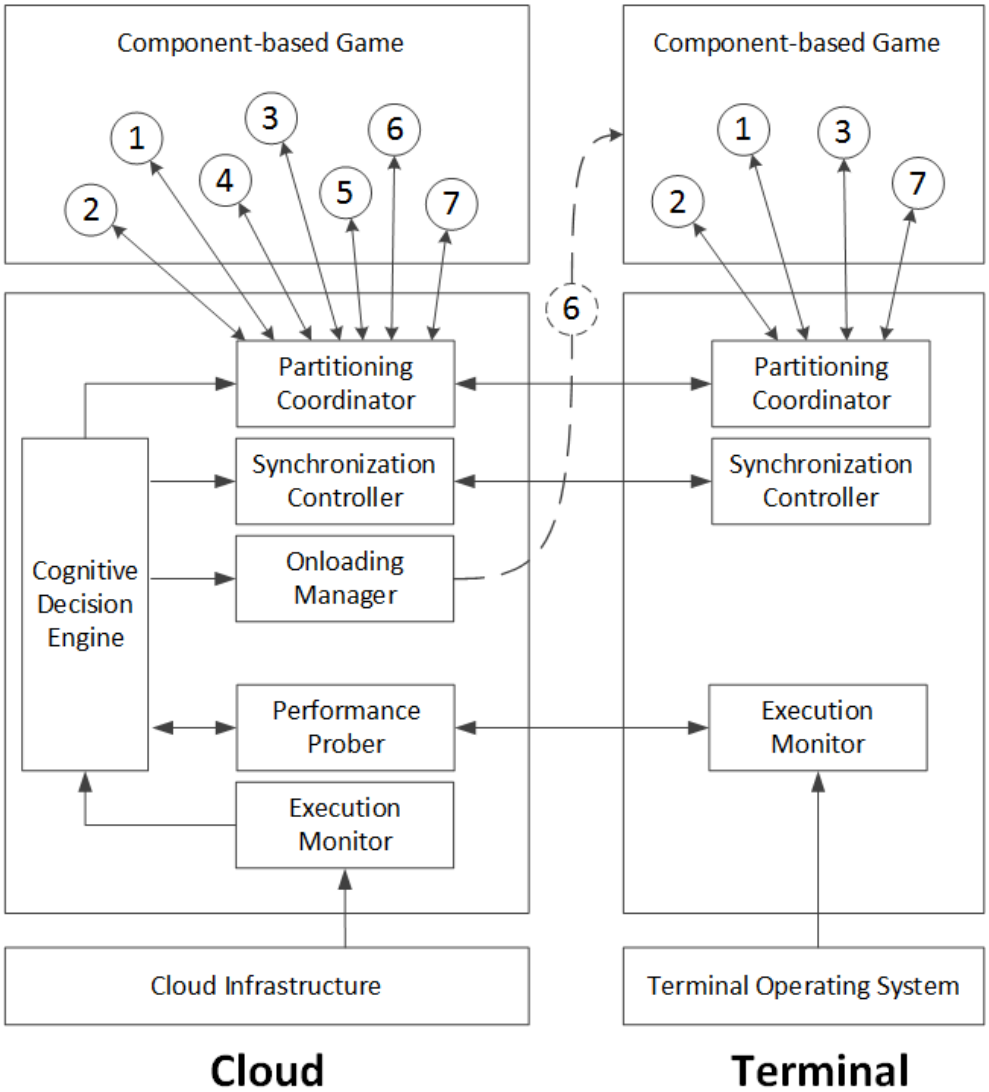


Figure 4.8: Decomposed Ubiquitous Cloud Gaming Platform

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 the 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.

4.4.1 Execution Monitor

One of the most critical problems for the decomposed cloud gaming platform is to design a practical mechanism to measure the execution status, e.g., component execution costs, execution probability, and communication costs. In our design, we introduce a *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 component, 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, execution environment, etc.

Table 4.2: Table of Execution Information

Component	**Memory**	**CPU**	**Time**	**Output**	...	**Environment**
3	22MB	5%	20ms	3KB	...	Terminal
1	42MB	1%	10ms	9KB	...	Cloud
5	2MB	0.3%	4ms	23KB	...	Terminal
4	7MB	3%	9ms	7KB	...	Terminal
1	42MB	1%	12ms	9KB	...	Cloud
...	...	...	...	...	...	...

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. Apparently, players with different gaming behavior might produce different invocation frequencies for these components. Hence, this database not only stores the measurement information from the perspective of cloud gaming system, but also facilitates proposed system's cognitive capability to learn the players' interactional behaviors.

4.4.2 Performance Prober

However, the information extracted from the execution information database is not sufficient for the proposed platform to perform cognitive adaption, since the system needs to evaluate the execution performance of each component both in cloud and terminal. This might not be possible in the 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 quality between the cloud and terminals. Therefore, we design a mobile agent [89] based Performance Prober to probe the Cloud-Pipe-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 report to the Performance Prober in the cloud. An illustration of mobile agent prober is depicted in Fig. 4.9. Note that, this

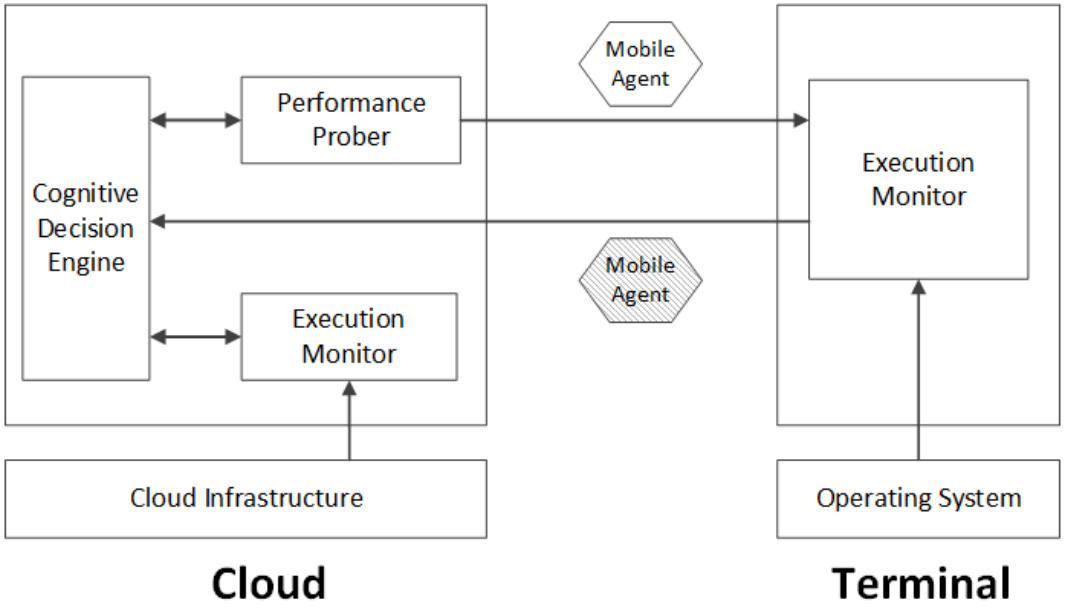


Figure 4.9: Mobile Agent Dispatch for Performance Prober

process involves two network transmissions, in which the system are able to calculate the network quality parameters, including, round-trip time and data transmission rate. With this approach, the Performance Prober is able to compare the computational efficiency of cloud and terminals, consequently, estimate the execution information for all components with computation. In our implementation, we denote the cloud computational efficiency as E_c, 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 $E_t \times R_E$.

In order to minimize the overhead of probing, the Performance Prober is designed to work collaboratively with Execution Monitor. As mentioned in Section 4.4.1, the cognitive system calculates the execution probability of each component in a statistical approach. Hence, a traverse in both cloud and terminals' database is necessary. Meanwhile, the statistics in the terminal should be reported to the cloud periodically. Accordingly, the

mobile agent component in Performance Prober is designed as the database information retriever and carrier to improve the system efficiency.

In our implementation, the system dispatches Performance Prober to the terminals in an interval of T_I and save the probing results in a database. Table 4.3 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 variety of the terminal, e.g., the network quality.

Table 4.3: Table of Performance Information

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

4.4.3 Cognitive Decision Engine

Cognitive Decision Engine is a 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 Onloading Manager, dynamic partitioning for Partitioning Coordinator and data synchronization for Synchronization Controller. The decision making for the engine is discussed in details in Chapter 3.

4.4.4 Onloading Manager

Since the cognitive platform supports *click-and-play*, none of the game components exists in the mobile client at the beginning of a gaming session. In this case, the cloud server

should be capable to transmit executable components to the mobile terminal, in order to enable dynamic resource allocation. The *Onloading Manager* employs the concept of the 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 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. In this context, the priority p_i for the ith game component is modeled by Equation 4.4. 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 mobile terminal.

$$p_i = f(r_i, c_i, f_i, \sum_j f_{t_{j,i}} \cdot t_{j,i}, , \sum_j f_{t_{i,j}} \cdot t_{i,j}) \tag{4.4}$$

To simplified the system, we implement three types of onloading in our platform: i) the system administrator is able to onload specific components to the client manually from the Cloud Configuration Center (as shown in Fig.4.11); ii) the platform is able to randomly dispatch selected components to the client, when the network connection between the cloud and mobile devices are idle; iii) the client is able to request and fetch 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, since the code length of the components are short, their transmission cost is negligible in the experimental results, we adopt the third mode.

4.4.5 Partitioning Coordinator

To facilitate dynamic partitioning, a flexible invocation routing mechanism should be implemented in the proposed platform. In other words, the proposed system should support component positioning and transparent message forwarding for components. In our design, all input and output data from the components are sent to the pair of *Partitioning Coordinator*s, which provide routing service for all invoke messages by intelligently selecting the destination components.

With the information from Execution Monitor and Performance Prober, the Cognitive Decision Engine construct a component invocation graph, as depicted in Fig. 3.3. In order to provide a QoE-oriented resource optimization, the dynamic partitioning problem intrinsically seeks to find a cut in the component invocation graph such that some components of the game execute on the client side and the remaining ones on the cloud side. The optimal cut maximizes or minimizes an objective function O, which expresses the general goal of a partition, e.g., minimizing the end-to-end interaction time between the mobile terminal and the cloud, minimizing the amount of exchanged data, or minimizing the latency in a gaming session.

In designing the objective function O, we need to satisfy the user's QoE requirement, including resource constraints in mobile devices and tolerable latency for each interaction cycle:

$$\sum_{k \in mobile} r_k \leq R_{Mobile} \tag{4.5}$$

Denote components set S_c involved in each gaming interaction cycle, for all $i, j \in S_c$,

$$\sum_{i \in mobile, j \in cloud} \frac{(f_{t_{j,i}} \cdot t_{j,i} + f_{t_{i,j}} \cdot t_{i,j})}{B} + T_{S_c} \leq T_m \tag{4.6}$$

where R_{Mobile} represents the available resource in the mobile device, T_{S_c} denotes the transaction delay of the components, B denotes the average bandwidth, and T_m denotes the average maximum delay that the players can tolerate.

Table 4.4: Cost Table of Candidate Solutions

Cloud	**Terminal**	**Bandwidth**	**Memory**	**CPU**	**Latency**
1,2,3	4,5,6,7	30kbps	102MB	12%	89ms
1,2,4,5	3,6,7	60kbps	230Mb	25%	142ms
1,3,4,5,6,7	2	10kbps	150Mb	17%	105ms
...	...	...	...	...	

In our implementation, the platform enumerates all possible partitioning solutions for the involving components and estimate their overall resource costs, including bandwidth consumption, memory usage, CPU percentage, overall latency, etc. Based on the estimation, the platform creates a cost table of candidate solutions, from which the optimal solution is determined. Table 4.4 shows an example of cost table of candidate solutions for 7 components.

4.4.6 Synchronization Controller

The remote distribution of game components results in the data synchronization problem. To address this problem, the *Synchronization Controller* is employed to update all parameters in the gaming environment. However, the synchronization process also introduces a non-negligible network overhead. Consequently, we design the synchronizing mechanism following the principle of "Sync-Only-If-Necessary" to minimize the transmission cost.

4.5 Testbed Implementation

In order to validate our proposal, we implement a ubiquitous cloud gaming testbed (UBCG Testbed) and develop game prototypes on this testbed. In this section, we discuss the implementation issues and our solutions.

4.5.1 Enabling Technologies

To implement the proposed cognitive platform for mobile cloud games, we seek to enable 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[10] 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, both server-side and client-side. This feature facilitates the game components, JavaScript gaming code in this context, to migrate from the cloud to user-end, and to be executed on cloud server and client as a mobile agent.

For the mobile client, we embed a WebKit-based browser into the cognitive engine for parsing and executing the JavaScript mobile agent from the cloud server. In our implementation, the WebKit browser is built on Android smartphone. However, all mobile operating systems supporting browsers are able to run our cognitive platform after a small number of modifications. We are also looking for alternative solutions to

[10] http://nodejs.org/

implement the mobile 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.

4.5.2 Deployment Directory

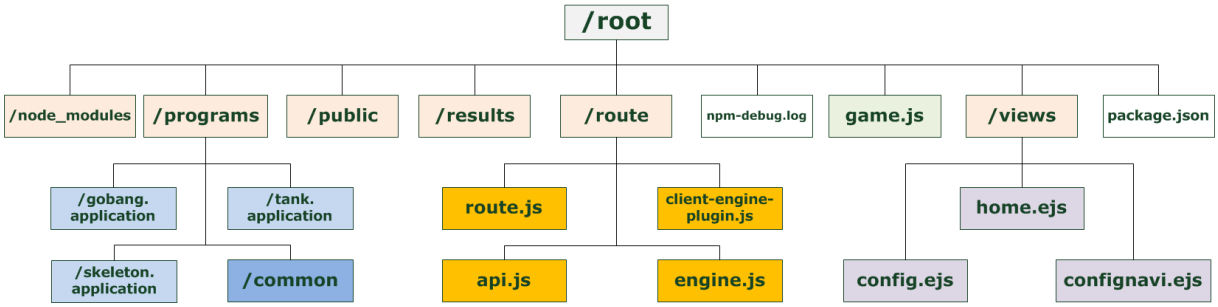


Figure 4.10: Deployment Directory of UBCG Testbed

As depicted in Fig. 4.10, besides the standard elements of Node.js project (*package.json*, *npm-debug.log* and *node_modules* directory that imports dependent Node.js libraries), the UBCG Testbed project is organized by Express[11], a Node.js web application framework following Model-View-Controller (MVC) software architectural pattern. Following is the deployment of our testbed:

- *game.js*: as the entrance of UBCG Testbed, *game.js* creates and launches a Node.js web server that listens to players' commands and responses corresponding messages or documents to them.

- */views*: as the portal of UBCG Testbed, Embedded JavaScript (EJS) template files in directory of *views* provide views for the game server, including *home.ejs*, *confignavi.ejs* and *config.ejs*. The *home.ejs* lists existing games on the testbed, so that the players can click through the icons to access the game.

[11] http://expressjs.com/

- */route*: as the testbed's core, the controllers and models designed in the directory of *route* enable the proposed dynamic partitioning. While *route.js* functions as the entrance to different views and *api.js* interprets the API invocations in applications, the combination of *engine.js* and *client-engine-plugin.js* are in charge of inter-component message redirection, following the partitioning decisions made by their built-in algorithms.

- */public*: as the container of plug-ins and accessaries, the directory of *public* loads all client-side third-party Cascading Style Sheets (CSS) and JavaScript files to players' terminal. Since our target is to develop a responsive, mobile first gaming environment, all layout designs adopt the Metro UI CSS[12], which is a set of Windows 8 template extended from Bootstrap[13] framework.

- */application*: for our application developers, their applications should contains at least one EJS view *app.ejs* and its accessorial components. Their codes will be deployed to the directory of *programs*, with name extensions of *.application* (e.g., *tank.application*). Note that, the components within directory of *common* can be accessed by all applications.

- */results*: as a full-access directory, *results* enables the application developers to write intermediate data into binary files. In our following experiments, we saves all resulting data produced by prototype programs here.

4.5.3 Application Programming Interface

As a testbed that supports component-based games, UBCG testbed provides a set of APIs to encapsulate lower layer partitioning for game developers. To acknowledge a

[12]http://metroui.org.ua/
[13]http://getbootstrap.com/

inter-component invocation in UBCG Testbed, the developers should simply add a "$$" mark before the name of the components when they are invoked in the code (e.g., $$$componentX(\{msg : args\})$; stands for an invocation of *componentX* with a parameter of *args* passing as a message).

4.5.4 The Administration Center

Figure 4.11: Screenshot of Administration Center of UBCG Testbed

Fig. 4.11 illustrates a screenshot of the UBCG Testbed administration center (rendered by config.ejs). The administrator can browse all ongoing gaming sessions here from the *TERMINALS* list session. For each terminal, the partitioning and loading

status of all components are illustrated in graphical user interface. Note that, if the *AUTO OPTIMIZATION SWITCH* turns to *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.

4.6 Prototype Development and Experiments

To demonstrate the feasibility and efficiency of UBCG testbed, we develop and deploy three prototypes in this section, including a Gobang game, a 3D skeletal game engine, and a Robocode tank game.

4.6.1 Development Challenges

We first discuss the challenges in developing decomposed game prototypes.

Global Variable

Global variables commonly exists in program design, as they can be accessed anytime anywhere while the program is running. However, the nature of component-based platform requires the system should be decomposed into independent components. In our implementation, they are basically JavaScript functions. In this case, once a component completes its life cycle and terminates its execution by invoking other components, all variables defined within its scope will be destroyed. In order to facilitate global variable, we investigate two different solutions.

The first approach is to save the variable into a JavaScript object named *global*, of which the lifetime is as long as the program. Because *global* is a JavaScript object itself,

then without any doubts that it can save any JavaScript variables in it. However, this object can not be existing on both cloud and terminals, which means the variables defined in *global* by a component executing on the client couldn't be accessed by a component executing on the server. Moreover, if the component migrates from terminal to server or reverse, all the variables saved in *global* will not be available, which leads the system to crash. To this end, JavaScript object *global* can be utilized to cover all kinds of JavaScript variables, while its global variables can only be shared among components in either client side or server side, not both.

The second solution is to save the variable into a JSON object named *context* and utilize the object as an argument being passed to the next component when one component invokes another. This approach, in fact, localizes the global variables as parameters, thus, solves the problem of sharing variables between cloud and terminals. However, it also has two unavoidable disadvantages: i) The *context* object can only carry JSON variables. Those user-defined JavaScript variables containing references to other variables or function objects cannot be saved into *context* as they are not JSON object. ii) The *context* object carries all global variables in all component invocations, while only a small proportion of them are accessed in most of the cases. This introduces remarkable overheads on network transmission when the object are passed between cloud and terminals.

In our development, we adopt a hybrid solution: some static global variables are implemented in JavaScript object *global*, while the others are converted into JSON objects for *context* transfer. This feature makes it difficult for us to employ third-party libraries to our prototypes. Hence, we give up the JavaScript physics engine OimoJS and CannonJS, since they all define a large number of non-JSON variables. Instead, we only utilize a small proportion of JavaScript render engine ThreeJS for basic rendering, while most of the rendering and simplified animation algorithms are implemented by ourselves,

where we adopt *context* as variables.

For the future improvement, we are targeting a synchronization mechanism for a *global* object, which updates themselves between cloud and terminals. However, this approach requires a novel system performance evaluation method, so that cognitive engine can still work. Another direction is to investigate the solution that encapsulates arguments transmission other than JSON object, which makes our platform better compatible to the third-party libraries. We are also considering a compress and decompress mechanism to minimize the data size in object transfer. This will reduce the burden of network transmission, while introducing more computational cost for compression and decompression.

Synchronization of Input and Output (I/O) States

Most browser-based programs handle I/O with a local interrupt mechanism. Once the user triggers a mouse or keyboard event, there will come up an interrupt which will be responded by the browser by calling corresponding predefined processing functions. As our prototypes are designed as component-based, the global variables are conveyed among components, which means that one single I/O event may need multiple handling functions in different components. Nevertheless, due to the asynchronization between the I/O event and the main control flow of the program, there might be a data hazard called WAR (Write after Read) as it is in computer architecture. For instance, our prototype is running a rendering component to calculate parameters, e.g., normal vectors, when the user drags his/her mouse to change the viewpoint, then I/O component will be triggered to receive and respond I/O event by saving the new state into his own *context* object. Afterward, the rendering component finished its execution and transfer the control flow back to I/O component with its out-of-date *context* object. In this case, the mouse dragging event will be ignored thoroughly without any handling, which basically is the WAR data hazard.

To avoid WAR data hazard, our solution is to save the I/O state information into *global* object when an I/O event comes up, since the *global* object won't be destroyed while control transferring between components. Next, right after the control transferred back to I/O component, we copy the I/O state information in *global* object into I/O component's current *context* object, then these I/O state information can be transferred forward to the following components to trigger the processing functions, which also makes I/O handling process synchronized with the execution of the program.

But this is not good enough since our I/O should only be processed once each time the event comes up while our handling functions are examining the I/O state information to process in every main loop execution. So we should reset the I/O state variables in *context* object every time an event has been handled, because handling functions are everywhere and their *global* objects are not synchronized. But these reset states will be overwritten by the *global* object in I/O component due to our above mechanism to avoid WAR. So the final solution is to create an additional property *IOSynch* in *context* object to signify whether the I/O event has just been handled. Every time when an I/O event is handling, we set the bit of this I/O state to 1. When the I/O component takes over the control flow, we first examine every *IOSynch*: If it is set to 0, then we copy this I/O state in *global* object into *context* object, while if it is 1, we reset its value to 0.

4.6.2 Gobang Game

Gobang game (also known as Gomoku or Five in a Row) is an abstract strategy board game that players alternate in placing a checker of their color on an empty intersection of the chessboard. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally. We developed the Gobang game prototype for UBCG Testbed to demonstrate the efficiency of offloading game engine's computational complexity, artificial

intelligence (AI) in this context, to the cloud. Therefore, we implement the AI module as a component, which is feasible to migrate between cloud and players' terminals and execute in these two different environments. A screenshot for developed Go Bang game is illustrated in Fig. 4.12;

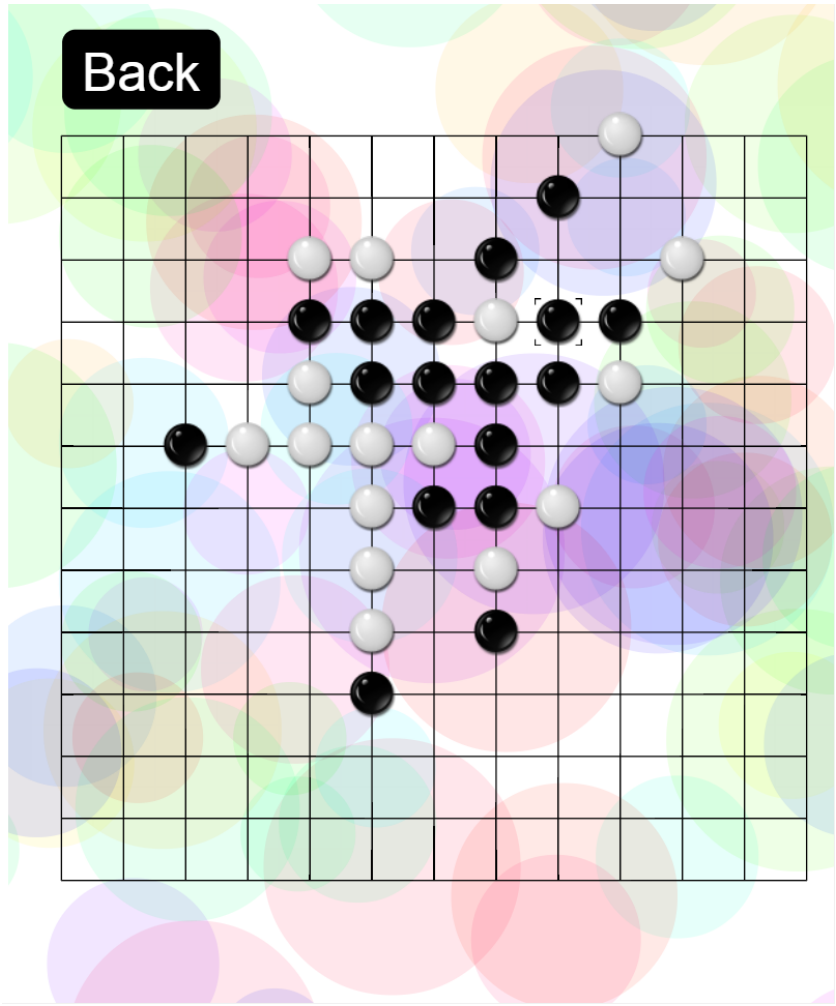


Figure 4.12: Screenshot for Gobang Game

Three types of devices are employed in our evaluation, including an ASUS windows 7 personal computer (PC) with Intel Pentium G630 @2.70GHz CPU and 4.0 GB Internal Memory (RAM), an Apple iPad mini tablet with 1 GHz dual-core ARM Cortex-A9 CPU and 512 MB DDR2 RAM, and a LG G2 mobile phone with 2.26 GHz quad-core Snapdragon

800 processor, 2.0 GB RAM and Long-Term Evolution (LTE) networks module. Through public Wi-Fi network at UBC Vancouver campus and Fido LTE cellular data network service in Vancouver, these devices are utilized as players' terminals to access the Gobang game deployed on UBCG Testbed hosting in Amazon Elastic Compute Cloud (EC2)[14].

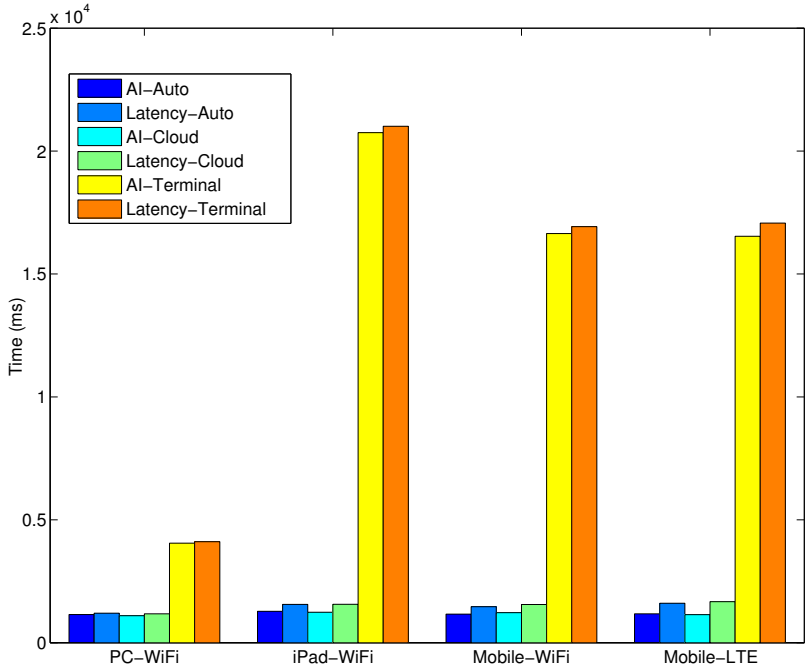


Figure 4.13: Response Latency Comparison in Gobang Game Prototype

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 (Testbed automatic optimization, all cloud execution and all terminal execution) and record two critical data: 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 and the time the AI placing a

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

chess. These measurements are depicted as six schemes in Fig. 4.13.

Apparently, the numeric value of AI-Auto is closed to Latency-Auto, since their differences are caused by two-time network communications between terminals and the cloud. According to our measurement, WiFi and LTE network introduce additional 344.37 ms and 485.25 ms delay in average, respectively. These delays are negligible in these experiments. This is the reason that Mobile-WiFi exhibits nearly identical pattern to Mobile-LTE. The most remarkable phenomenon is that the cloud schemes reduce a huge proportion of response time in comparison to terminal schemes, this indicates the high computational complexity of designed AI components. Apparently, the AI component's feature of high resource consumption makes it better to be executed in the powerful cloud. This conclusion is proved by another observation in our experimental series: all automatic optimization solutions choose to do this, resulting comparable performances between Auto and Cloud solutions.

4.6.3 3D Skeletal Game Engine

The 3D skeletal engine, our second prototype, aims to challenge UBCG Testbed's capacity on rendering 3D game scenes. Besides the complexity of AI, modern games are also prone to create fantastic game scenes that consume a huge amount of terminal resources. Recent work [49] has explored the possibility of partial offloading for game scene renderings. The 3D skeletal engine is our understanding in this perspective. A 3D skeletal system (also know as the bone system) is a common technique used to create skeletal animations in video games. As the foundation of generating acting units, a skeletal animation consists of a skin mesh and an associated bone structure, so that the movement of the mesh is associated with the vertices of the bone, following an exactly same pattern in reality: human beings have a skeletal structure covering with muscles and skin on it. The developed 3D Skeletal

Game Engine consists of an animation editor and a 3D rendering module, which computes and draws action animations for a human and a dog, respectively. With Separate Skinning method and Forward Kinematics in OpenGL basic bone system[15], the implementation screenshot of a four-component prototype is illustrated in Fig. 4.14.

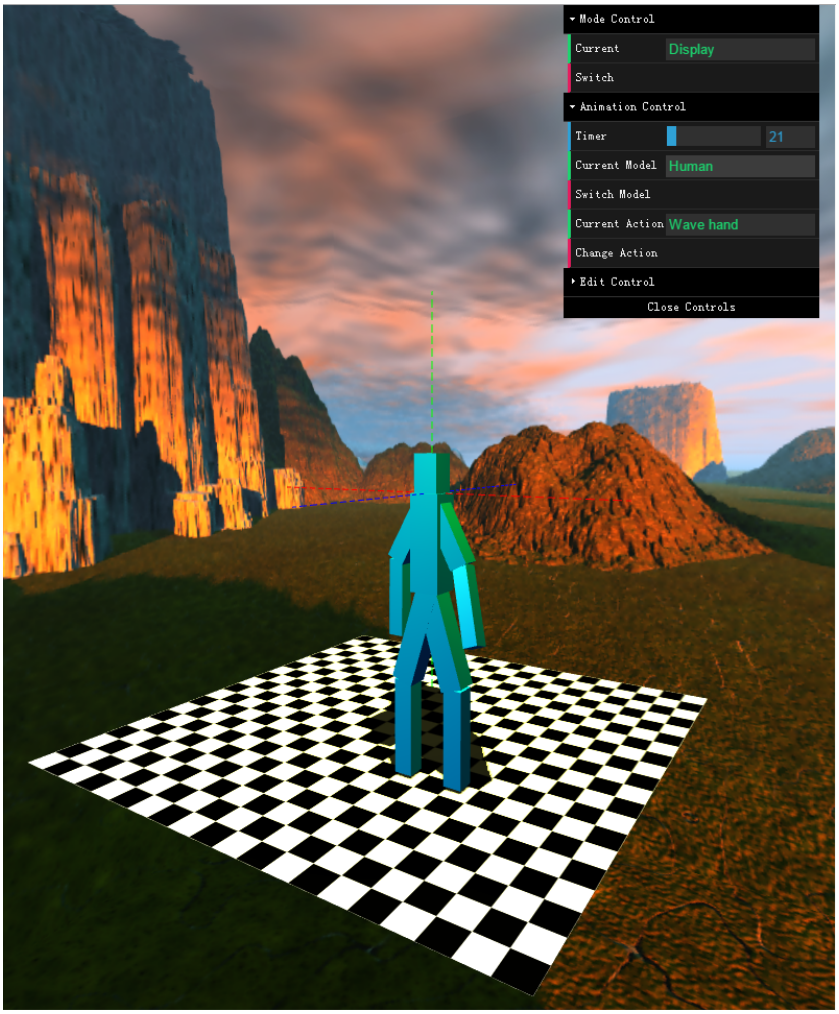


Figure 4.14: Screenshot for 3D Skeletal Game Engine

To validate and demonstrate the UBCG Testbed's feature of cognitive task allocation to the network quality, we conduct experiments to measure the fluency of rendered animations by the numeric value of frame per second (FPS). In order to explicitly control the network

[15]http://content.gpwiki.org/OpenGL:Tutorials:Basic_Bones_System

parameters between the cloud and terminals, we employ two identical computers to serve as cloud and client, which are equipped with Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz processor, 8.00 GB installed memory (RAM) and 64-bit Windows 7 operating system. On the "cloud" side, we installed NetBalancer[16] 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 should be comparisons between automatic optimization and all potential partitioning schemes. Since 3D 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 KB/S, 500 KB/S and 100 KB/S, respectively.

Fig.4.15 shows the results of above experiments. The average FPS of each time slot (5 seconds per slot) indicates the fluency of rendered animation at the specific time period. We can conclude from the comparison between different bandwidth schemes that, network quality plays an important role in the proposed real-time rendering prototype. The resulting FPS decrease from around 48 to around 18, when the bandwidth falls from 1000 KB/S to 500 KB/S. Things get worse if the network bandwidth is reduced to 100 KB/S, UBCG Testbed can only render the 3D skeleton at FPS rate of 5, which is only 10% of the 1000KB/S case. The good part of this experiment is that, it proves the efficiency of UBCG Testbed. Under all three network conditions, the cognitive engine does a great job in seeking optimal partitioning solutions for the prototype: Auto series 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, 16th

[16]https://netbalancer.com/

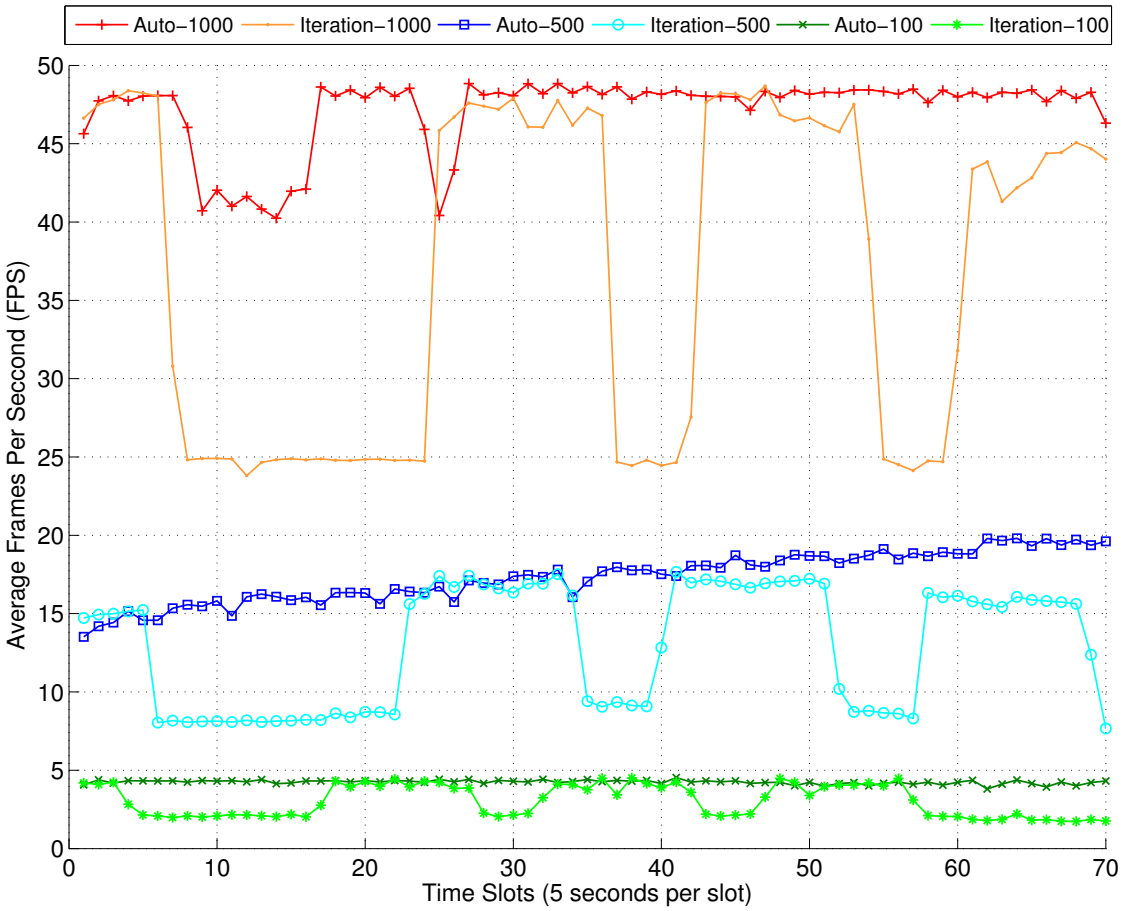


Figure 4.15: Game QoS (FPS) Comparison in 3D Skeletal Game Engine

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.

4.6.4 Robocode Tank Game

The idea of the third game prototype, Robocode Tank, comes from a famous open source educational game Robocode[17], 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. Our Robocode Tank

[17]http://robocode.sourceforge.net/

game prototype inherits all features of Robocode and places an additional tank controlled by players into the battlefield. Screenshot for implemented tank game is illustrated in Fig. 4.16.

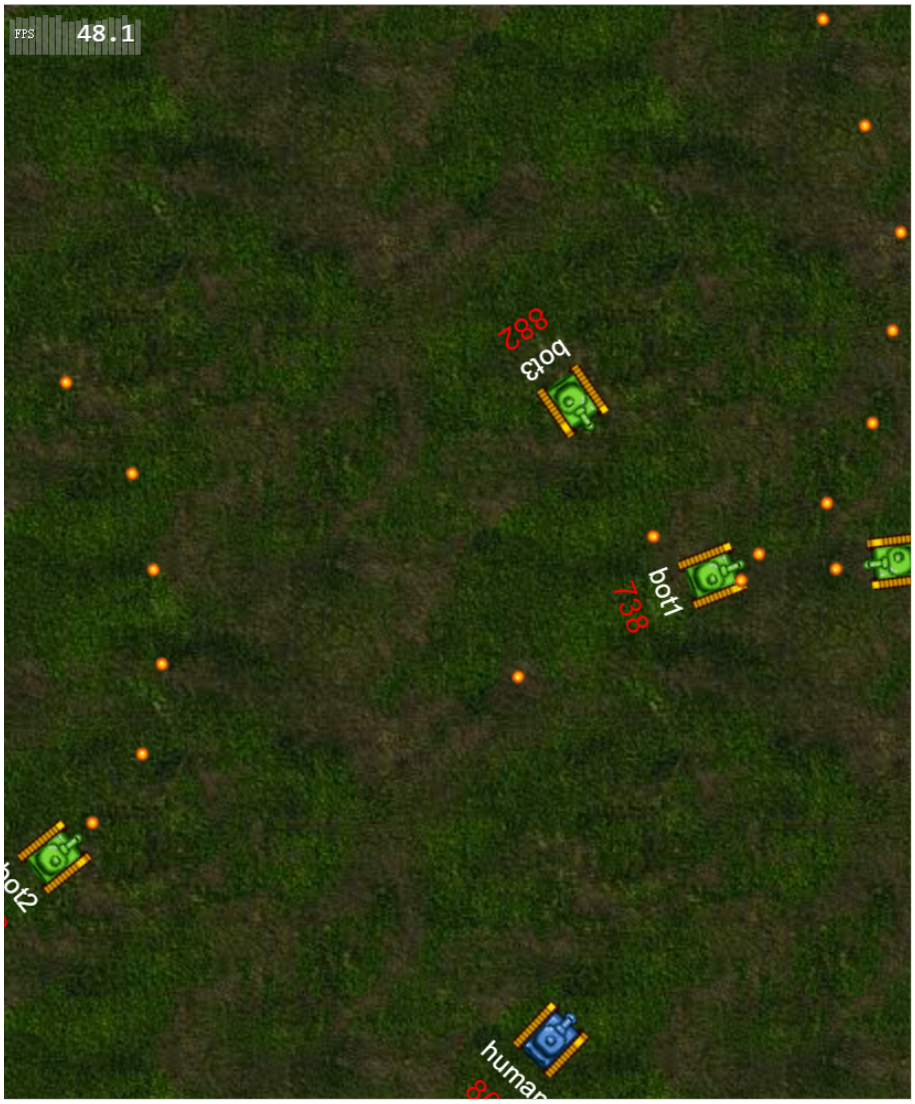


Figure 4.16: Screenshot for Robocode Tank Game

Since the system performance of Robocode tank game is subject to the complexities of tank AIs, we only demonstrate the cognitive capacity of UBCG Testbed with 3 gaming sessions. We create 4 AI controlled tanks as four components, running on the same canvas for competition. Each tank conducts itinerary planning algorithm with random

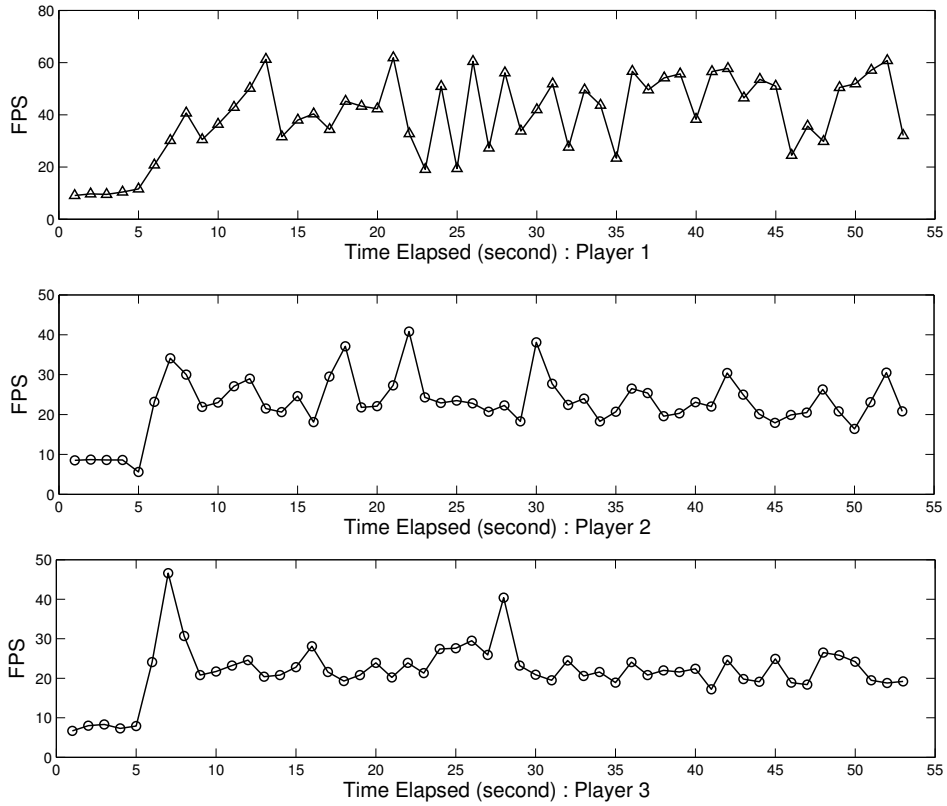


Figure 4.17: Game QoS (FPS) Enhancement for Robocode Tank Game

computational load, which is simulated by a loop repeating from 1 to 10000 times with uniform distribution. In our experiment, UBCG testbed is hosted in the EC2 Ubuntu Linux instance at us-west-2a zone, accessing through UBC Vancouver campus WiFi network. All of these three gaming sessions are conducted in a Google Chrome browser on a Windows 2012R2 PC with Intel Core i5-4670 @3.40GHz CPU and 6.0 GB Internal Memory (RAM). As shown in Fig. 4.17, these players all experienced very low FPS rate at the beginning of the gaming sessions, while the testbed eventually provided an optimal partitioning solutions for them (around 60 FPS). Note that, the solution search time for these three players are distinct from each other, because of the ever-changing real-time system status.

4.7 Summary

In this chapter, we have proposed a cognitive, flexible and promising gaming platform for mobile cloud gaming, which supports click-and-play, intelligent resource allocation and partial offline execution. 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, synchronization and redirection services for partial offline execution. We discussed the enabling technology and implemented the proposed platform as a pure JavaScript solution. Extensive experiments were performed to show that the adaptive partitioning is able to provide an optimal solution in terms of overall latency. For the future work, we focus on the following directions: i) To better scheduling and resource management, we need to measure the computational performance of game components, both in the cloud and mobile devices. In addition,the communication between these components should also be measured. ii) The platform should accurately, timely and efficiently measure, evaluate and predict the real-time system environment, to provide a reference for QoE-oriented adaption. iii) Instead of overall latency, we should consider more sophistic models with more impact factors, such as computational performance, network bandwidth, battery percentage, etc.

Chapter 5

Conclusions and Future Work

In this chapter, we conclude the presented works in this thesis and also suggest several topics for future work.

5.1 Conclusions

- In Chapter 2, we have investigated a novel cloud gaming framework in which multiple players in the same game scene can share their video frames with each other. It is facilitated by an ad hoc cloudlet, which consists of a secondary P2P network. We have investigated the feasibility of video sharing by modeling the correlations among video frames from different players and the avatars' behaviors in the gaming sessions. Furthermore, we have explicitly formulated the proposed system, including mobility of terminal devices and diversity of network quality of service for distinct players. Optimal encoding schemes and heuristic algorithms that derive scalable solutions have been introduced in our work. An empirical study on Diablo II and extensive trace-driven simulation results have been presented to demonstrate the effectiveness of the proposed QoE optimization scheme and the efficiency of the heuristic algorithms. This study has demonstrated the potential of optimizing players' QoE by exploring correlations among peer devices' gaming videos. For future research, the correlations among videos from different game genres, e.g., the first-persons and the second-persons, should be investigated. In

addition, improving the encoding efficiency in the cloud and the decoding capacity in the terminals are also critical challenges.

- In Chapter 3, we have presented the idea of decomposed cloud gaming systems and modeled the cognitive resource management for decomposed cloud gaming. As the core of the proposed system, the resource management issues were formulated as an optimal cut problem in a weighted graph. By theoretical analysis and simulations, we have demonstrated the flexibility of the proposed system over diverse optimization targets. In order to efficiently derive the optimal solutions with different optimization targets, two heuristic algorithms have been proposed and evaluated. We have presented extensive simulations and testbed-based experimental results to show that the cognitive platform provides great efficiency in terms of resource minimization and throughput optimization while guaranteeing the QoS requirements for game sessions. We learned from this research that decomposition and dynamic partitioning methodologies can be applied to gaming programs. However, the inter-component communications in these programs should satisfy certain requirements. The next step of this study should be the empirical measurements of game software, emphasizing on the invocation features among independent modules.

- In Chapter 4, we have discussed the design issues in developing decomposed cloud gaming system and presented a design of a cognitive and flexible system for ubiquitous cloud gaming, which supports click-and-play, intelligent resource allocation and partial offline execution. 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, synchronization and redirection services for partial offline execution.

We discussed the enabling technology and implemented the very first testbed in a pure JavaScript solution. Three prototype games have been developed to validate the feasibility and efficiency of the proposed decomposed cloud gaming system. One of the most important lessons we learnt from this study is that there are still a number of difficulties in the development of decomposed software. Future work in this topic should investigate better encapsulation methods for independent modules.

5.2 Suggestions for Future Work

In the following, we consider several interesting possibilities for extension of the current work.

5.2.1 Cloud Gaming Engages More Multiplayer Games

The gaming industry has seen a shift towards games with multiplayer facilities. A larger percentage of games on all types of platforms include some form of competitive online multiplayer capability. However, researchers in cloud gaming have yet to explore the huge potential of cloud gaming in multiplayer scenarios, which involve more than one players in the same game environment at the same time. In such scenarios, players can interact with each other in partnership, competition, or rivalry, which provide them with opportunities for social communication that is absent in single-player games. We define multiplayer games as either massively multiplayer online role-playing games (MMORPGs) (e.g., World of Warcraft, Lineage) or small-scale networked games (e.g., StarCraft, Diablo, League of Legends). Due to the rapid developments in both computer and broadband network technologies, MMORPGs have become a very important part of the online entertainment industry nowadays. Game players tend to get connected to play games with a group of peers instead of against artificial intelligence. This change enhances the gameplay experience and

makes more players become attached to this type of entertainment. Similarly, small-scale networked games are also attracting many players, due to their flexibility to form groups to conduct short sessions of gaming. In fact, applying cloud gaming concept to multiplayer gaming introduces additional benefits as follows:

1. **Nature of Connectivity:** A critical drawback of the cloud gaming paradigm is the indispensable network connectivity. Indeed, an overhead is incurred to establish and maintain the network connections between the cloud and players terminals during the gaming session. This may be a reason that has kept customers away from cloud gaming. However, this worry is unlikely to impact the decisions of end users when it comes to multiplayer games, since network access is already mandatory for such games.

2. **Temporary Engagement:** An important feature of cloud gaming is to enable game play without download and installation. This nature of click-and-play becomes more attractive in a multiplayer scenario where people in proximity are engaged to play the same game temporarily. For instance, several friends in a party might decide to play some video games together but they could not find a game that is installed in all their smartphones. In this case, the benefit of click-and-play cloud gaming becomes self-evident.

3. **Gaming Fairness:** How to achieve fairness between multiplayer is a crucial issue in designing online games. As the players are competing with each other in the same game scene, the system should respond to their actions immediately. Players of a conventional online game may suffer from unfairness, especially when the QoS (e.g., latency, packet loss rate) of their network connections varies. With cloud gaming, players gaming instances are hosted in the cloud. Hence, the message exchanges between game instances occur inside the cloud, which makes it easier to

maintain a guaranteed QoS. The cloud gaming system may be able to adapt itself to a terminal's network to provide better fairness. For example, previous research proposed to adjust rendering parameters to reduce video quality for those players with poor network access. By reducing the video quality, players with less capable devices or experiencing poor network conditions can be treated fairly in multiplayer games.

Research challenges include: **i) Video Sharing:** Video sharing cooperative cloud gaming reduces bandwidth consumption with cooperative encoding. However, it also brings several challenges and research issues. First, introducing reference-based encoding brings additional overhead to the system, such as increased workload in the video encoder servers. A common assumption in cloud computing paradigm is that all game engines and video encoder server are extremely powerful to perform the encoding, due to scalable computing resources. However, the cost of cloud resources cannot be neglected in practice. Hence, optimization inside the cloud should be considered. Second, decoding video frames from predicted images require additional cloudlet support, or using an ad hoc networking model that enables terminals to decode the videos cooperatively. Energy consumption of the mobile terminals to perform the tasks of cooperative video decoding and ad hoc network communications can be a critical issue. Furthermore, system performance in the presence of device mobility could also be an important issue. **ii) Cooperative Component Sharing:** Component sharing cooperative cloud gaming should be built upon the concept of component-based gaming architecture. The most commonly seen challenge for such an architecture is the decomposition complexity, or, to be more specific, the decomposition level (e.g., data level, task level, function level). The decomposition level defines the frequency that components interact with each other, and thus the rate of data exchange between components. It is actually the determining factor

in the ad-hoc cloudlet based gaming architecture. Since components could be remotely executed, a high data exchange rate (high decomposition level) between remote components could be highly detrimental to both the system performance and communication cost. As the decomposition level varies with game genre, how to find the appropriate level of decomposition remains the biggest challenge. Furthermore, the beacon messages and the memory used to acquire and store the neighbors gaming statuses are overheads that require further modeling and analysis. Moreover, efficient and decentralized service discovery, device discovery, and membership management mechanisms should be carefully designed to ensure the scalability of the system.

5.2.2 Novel Gaming Paradigm Convergence in Cloud

Cloud computing provides additional opportunities for novel games paradigms, such as virtual reality (VR) games, augmented-reality games and context-aware games.

1. **Virtual Reality Cloud Gaming:** VR games have been talked about for years, but we have yet to see many of them become availble to the market. A number of leading companies, such as Oculus, Valve and HTC, are on their way to produce high-quality equipment to facilitate VR games. If realized, the industry then needs to start building content for this new and potentially "game changing" platform as quickly as possible. However, the real-time rendering of omnipresent 3D scenes requires very strong graphical computation power, which might limit the application of VR games. A potential solution involves the cloud, as it provides rich resource up in the clusters, e.g., NVIDIA GRID, etc. The high volume of memories and computational power make the infinite virtual world for the players become possible.

2. **Augmented-Reality Cloud Gaming:** Project Glass is a research and development program by Google to develop an augmented reality head-mounted

display [90]. In contrast to traditional mobile devices, Google Glasses provide a hands-free display of information on the lenses, integrating the virtual display to the reality in one's vision. In addition, the device enables people to interact with the Internet via natural language voice command. Therefore, it is a perfect solution for augmented-reality cloud games, which can be launched while people are walking. A typical augmented-reality cloud gaming can be demonstrated by the following examples. The camera on the glasses continuously captures the player's vision in real-time, and the device transmits the video to the cloud via a wireless network. In the cloud-end, the video analyzer processes the video images with sophisticated artificial intelligence technologies, such as pattern recognition. The game logic in the cloud then creates gaming contents and delivers them to the game players. These virtual gaming contents, such as coins and bombs, can be displayed in the real scenarios through the lenses. Therefore, the system provides the players a gaming world with mixed virtual and real items. During the gaming session, the players should move their bodies or their vision angles to interact with those virtual items, in order to achieve the designed gaming goals. This type of games can be used in daily exercise and for health-care purposes. However, how to guarantee the safety of players during the gaming session remains a critical issue for game designers.

3. **Context-Aware Cloud Gaming:** An example of context-aware cloud gaming is gaming onboard a vehicle. People nowadays prefer to entertain themselves with games when they are trying to pass time onboard a bus or subway train. The mobility of a vehicle provides a new gaming experience for players. In this gaming scenario, the vehicular game reports Global Positioning System (GPS) information to the cloud via a wireless network, so that the cloud is able to deliver corresponding gaming

contents to the mobile devices. For example, when the player is in the urban area, the environment of the game is set to be in the crowd and is busy; when the player is in the suburb area, virtual wild animals might appear in the game and attack the avatar. Furthermore, the game also collects the mobility information with its equipped accelerators and the cloud utilizes these sensed data to facilitate various gaming contents. For example, when the vehicle is accelerating, the avatar in the game will enter a speed-up mode, such that the player has less response time to deal with the challenges in the game. In addition, with the assistance of the cloud, the game is able to search for peer players, e.g., those in the same vehicle, thus introducing more interactive gaming scenarios such as encountered challenges.

Bibliography

[1] P. Ross, "Cloud computing's killer app: Gaming," *IEEE Spectrum*, vol. 46, no. 3, pp. 14–14, 2009.

[2] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *Communications Magazine, IEEE*, vol. 46, no. 1, pp. 56–63, 2008.

[3] S. Wang and S. Dey, "Modeling and characterizing user experience in a cloud server based mobile gaming approach," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, Nov.-Dec. 2009, pp. 1 –7.

[4] M. Chen, "Amvsc: A framework of adaptive mobile video streaming in the cloud," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, 2012, pp. 2042–2047.

[5] S. Wang and S. Dey, "Rendering adaptation to address communication and computation constraints in cloud mobile gaming," in *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, USA, 2010, pp. 1–6.

[6] K. Chen, Y. Chang, P. Tseng, C. Huang, and C. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM international conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 1269–1272.

[7] J. Vanhatupa, "Browser games: The new frontier of social gaming," in *Recent Trends in Wireless and Mobile Networks*, ser. Communications in Computer and Information Science, vol. 84. Springer Berlin Heidelberg, 2010, pp. 349–355.

[8] W. Cai, M. Chen, and V. Leung, "Toward gaming as a service," *IEEE Internet Computing*, pp. 12–18, May 2014.

[9] D. Mishra, M. E. Zarki, A. Erbad, C.-h. Hsu, N. Venkatasubramanian, and M. El Zarki, "Clouds + Games: A Multifaceted Approach," *Internet Computing, IEEE*, vol. 18, no. 3, pp. 20–27, May 2014.

[10] L. Riungu-kalliosaari, J. Kasurinen, and K. Smolander, "Cloud Services and Cloud Gaming in Game Development," *IADIS International Conference Game and Entertainment Technologies 2013 (GET 2013)*, no. 1, 2013.

[11] A. Ojala and P. Tyrvainen, "Developing cloud business models: A case study on cloud gaming," *Software, IEEE*, vol. 28, no. 4, pp. 42 –47, July-Aug. 2011.

[12] C. Moreno, N. Tizon, and M. Preda, "Mobile cloud convergence in gaas: A business model proposition," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, Jan. 2012, pp. 1344 –1352.

[13] S. Dey, "Cloud mobile media: Opportunities, challenges, and directions," in *2012 International Conference on Computing, Networking and Communications, ICNC'12*, 2012, pp. 929–933.

[14] O. Soliman, A. Rezgui, H. Soliman, and N. Manea, "Mobile Cloud Gaming: Issues and Challenges," in *10th International Conference, MobiWIS 2013, Paphos, Cyprus, August 26-29, 2013. Proceedings*, ser. Lecture Notes in Computer Science, F. Daniel, G. Papadopoulos, and P. Thiran, Eds. Springer Berlin Heidelberg, 2013, vol. 8093, pp. 121–128.

[15] Z. Wu, "Gaming in the cloud : one of the future entertainment," *Interactive Multimedia Conference 2014*, 2014.

[16] K.-t. Chen, C.-y. Huang, C.-h. Hsu, and G. Integration, "Cloud Gaming Onward: Research Opportunities and Outlook," *Proceeding of The 2014 IEEE International Conference on Multimedia and Expo (ICME2014)*, no. Proceeding of The 2014 IEEE International Conference on Multimedia and Expo (ICME2014), 2014.

[17] S.-P. CHUAH, C. YUEN, and N.-M. CHEUNG, "Cloud Gaming: A Green Solution to Massive Multiplayer Online Games," *IEEE Wireless Communications*, no. August, pp. 78–87, 2014.

[18] E. Depasquale, A. Zammit, M. Camilleri, S. Zammit, A. Muscat, P. Mallia, and S. Scerri, "An analytical method of assessment of RemoteFX as a cloud gaming platform," in *Mediterranean Electrotechnical Conference (MELECON), 2014 17th IEEE*, Apr. 2014, pp. 127–133.

[19] K. I. Kim, S. Y. Bae, D. C. Lee, C. S. Cho, H. J. Lee, and K. C. Lee, "Cloud-Based gaming service platform supporting multiple devices," *ETRI Journal*, vol. 35, pp. 960–968, 2013.

[20] C. Huang, K. Chen, D. Chen, H. Hsu, and C. Hsu, "GamingAnywhere: The First Open Source Cloud Gaming System," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, no. 1s, pp. 10:1—-10:25, Jan. 2014.

[21] 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 *Proceedings of the 19th ACM international conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 103–112.

[22] X. Nan, X. Guo, Y. Lu, Y. He, L. Guan, S. Li, and B. Guo, "A novel cloud gaming framework using joint video and graphics streaming," in *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, July 2014, pp. 1–6.

[23] K. Lee, D. Chu, E. Cuervo, A. Wolman, and J. Flinn, "Demo: DeLorean: Using Speculation to Enable Low-latency Continuous Interaction for Mobile Cloud Gaming," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: ACM, 2014, p. 347.

[24] M. Claypool, "Motion and scene complexity for streaming video games," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG '09. New York, NY, USA: ACM, 2009, pp. 34–41.

[25] M. Claypool, D. Finkel, A. Grant, and M. Solano, "Thin to win? network performance analysis of the onlive thin client game system," in *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, 2012, pp. 1–6.

[26] K.-t. Chen, Y.-c. Chang, H.-j. Hsu, and D.-y. Chen, "On the Quality of Service of Cloud Gaming Systems," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 480–495, Feb. 2014.

[27] M. Manzano, J. Hernandez, M. Uruena, and E. Calle, "An empirical study of cloud gaming," in *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, 2012, pp. 1–2.

[28] M. Manzano, M. Urueña, M. Sužnjević, E. Calle, J. A. Hernández, and M. Matijasevic, "Dissecting the protocol and network traffic of the OnLive cloud gaming platform," *Multimedia Systems*, vol. 20, no. 5, pp. 451–470, 2014.

[29] R. Shea, J. Liu, E. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *Network, IEEE*, vol. 27, no. 4, pp. –, 2013.

[30] R. Shea and J. Liu, "On GPU Pass-Through Performance for Cloud Gaming: Experiments and Analysis," in *Proceedings of Annual Workshop on Network and Systems Support for Games*, ser. NetGames '13. Piscataway, NJ, USA: IEEE Press, Dec. 2013, pp. 6:1—-6:6.

[31] H.-j. Hong, C.-r. Lee, K.-t. Chen, C.-y. Huang, and C.-h. Hsu, "GPU Consolidation for Cloud Games : Are We There Yet ?" *Network and Systems Support for Games (NetGames), 2014 13th Annual Workshop on*, 2014.

[32] M. Suznjevic, J. Beyer, L. Skorin-kapov, S. Moller, N. Sorsa, T. U. Berlin, and S. Juni, "Towards understanding the relationship between game type and network traffic for cloud gaming," in *Multimedia and Expo Workshops (ICME), 2014 IEEE International Conference on*, July 2014, pp. 1–6.

[33] U. Lampe, Q. Wu, S. Dargutev, R. Hans, A. Miede, R. Steinmetz, and U. L. B, "Assessing Latency in Cloud Gaming," *Cloud Computing and Services Science SE - 4*, vol. 1, pp. 52–68, 2014.

[34] Z. Xue, D. Wu, J. He, X. Hei, and Y. Liu, "Playing High-End Video Games in the Cloud: A Measurement Study," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 25, no. 12, pp. 2013–2025, Dec. 2015.

[35] Y. Chang, P. Tseng, K. Chen, and C. Lei, "Understanding the performance of thin-client gaming," in *Communications Quality and Reliability (CQR), 2011 IEEE International Workshop Technical Committee on*, 2011, pp. 1–6.

[36] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hossfeld, "An evaluation of qoe in cloud gaming based on subjective tests," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, June-July 2011, pp. 330 –335.

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

[38] S. Möller, D. Pommer, J. Beyer, J. Rake-revelant, T. I. Laboratories, and D. T. Ag, "Factors Influencing Gaming QoE : Lessons Learned from the Evaluation of Cloud Gaming Services."

[39] I. Slivar, M. Suznjevic, L. Skorin-kapov, and M. Matijasevic, "Empirical QoE Study of In-Home Streaming of Online Games," *Network and Systems Support for Games (NetGames), 2014 13th Annual Workshop on*, 2014.

[40] Y. Lee, K. Chen, H. Su, and C. Lei, "Are all games equally cloud-gaming-friendly? an electromyographic approach," in *Proceedings of IEEE/ACM NetGames 2012*, Oct. 2012.

[41] P. Quax, A. Beznosyk, W. Vanmontfort, R. Marx, and W. Lamotte, "An evaluation of the impact of game genre on user experience in cloud gaming," in *Games Innovation Conference (IGIC), 2013 IEEE International*, Sep. 2013, pp. 216–221.

[42] M. Claypool and D. Finkel, "The Effects of Latency on Player Performance in cloud-based games," *Network and Systems Support for Games (NetGames), 2014 13th Annual Workshop on*, 2014.

[43] K. Raaen, "Can gamers detect cloud delay ?" *Network and Systems Support for Games (NetGames), 2014 13th Annual Workshop on*, vol. 200, pp. 7–9, 2014.

[44] C.-y. Huang, C.-h. Hsu, D.-Y. Chen, and K.-T. Chen, "Quantifying User Satisfaction in Mobile Cloud Games," in *Proceedings of Workshop on Mobile Video Delivery*, ser. MoViD'14. New York, NY, USA: ACM, 2013, pp. 4:1—-4:6.

[45] S. Wang and S. Dey, "Cloud mobile gaming: modeling and measuring user experience in mobile wireless networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 1, pp. 10–21, July 2012.

[46] Y. Liu, S. Wang, and S. Dey, "Modeling, characterizing, and enhancing user experience in Cloud Mobile Rendering," *2012 International Conference on Computing, Networking and Communications (ICNC)*, pp. 739–745, Jan. 2012.

[47] L. Xu, X. Guo, Y. Lu, S. Li, O. C. Au, and L. Fang, "A low latency cloud gaming system using edge preserved image homography," in *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, July 2014, pp. 1–6.

[48] L. Lin, X. Liao, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li, "LiveRender: A Cloud Gaming System Based on Compressed Graphics Streaming," in *Proceedings of the ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 347–356.

[49] D. Meilander, F. Glinka, S. Gorlatch, L. Lin, W. Zhang, and X. Liao, "Bringing mobile online games to clouds," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, April 2014, pp. 340–345.

[50] 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 *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 2, Jan. 2006, pp. 994 – 997.

[51] J.-P. Laulajainen, T. Sutinen, and S. Jarvinen, "Experiments with qos-aware gaming-on-demand service," in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, vol. 1, April 2006, pp. 805 –810.

[52] S. Wang and S. Dey, "Addressing response time and video quality in remote server based internet mobile gaming," in *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, 2010, pp. 1–6.

[53] Y. Lu, Y. Liu, and S. Dey, "Enhancing Cloud Mobile 3D display gaming user experience by asymmetric graphics rendering," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*, Feb. 2014, pp. 368–374.

[54] L. He, G. Liu, and C. Yuchen, "Buffer status and content aware scheduling scheme for cloud gaming based on video streaming," in *Multimedia and Expo Workshops (ICME), 2014 IEEE International Conference on*, Jul. 2014, pp. 1–6.

[55] A. Bujari, M. Massaro, and C. E. Palazzi, "Vegas Over Access Point: Making Room for Thin Client Game Systems in a Wireless Home," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 25, no. 12, pp. 2002–2012, Dec. 2015.

[56] J. Wu, C. Yuen, N. Cheung, J. Chen, and C. W. Chen, "Enabling Adaptive High-Frame-Rate Video Streaming in Mobile Cloud Gaming Applications," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 25, no. 12, pp. 1988–2001, Dec. 2015.

[57] M. Hemmati, A. Javadtalab, A. A. Nazari Shirehjini, S. Shirmohammadi, and T. Arici, "Game as video: Bit rate reduction through adaptive object encoding," in *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '13. New York, NY, USA: ACM, 2013, pp. 7–12.

[58] G. Cheung, N.-M. Cheung, and A. Ortega, "Optimized frame structure using distributed source coding for interactive multiview streaming," in *IEEE International Conference on Image Processing*, Cairo, Egypt, Nov. 2009.

[59] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, ser. MCS '10. New York, NY, USA: ACM, 2010, pp. 6:1–6:5.

[60] M. Black and W. Edgar, "Exploring mobile devices as grid resources: Using an x86 virtual machine to run boinc on an iphone," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*, Oct. 2009, pp. 9 –16.

[61] E. E. Marinelli, "Hyrax : Cloud computing on mobile devices using mapreduce," *Science*, vol. 0389, no. September, 2009.

[62] P. Sharma, S.-J. Lee, J. Brassil, and K. Shin, "Aggregating bandwidth for multihomed mobile collaborative communities," in *IEEE Transactions on Mobile Computing*, vol. 6, no.3, Mar. 2007, pp. 280–296.

[63] R. Bhatia, L. E. Li, H. Luo, and R. Ramjee, "ICAM: Integrated cellular and ad hoc multicast," in *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, Aug. 2006, pp. 1004–1015.

[64] X. Liu, G. Cheung, and C.-N. Chuah, "Structured network coding and cooperative wireless ad-hoc peer-to-peer repair for WWAN video broadcast," in *IEEE Transactions on Multimedia*, vol. 11, no.4, 2009.

[65] M. Levoy and P. Hanrahan, "Light field rendering," in *ACM SIGGRAPH*, New Orleans, LA, Aug. 1996.

[66] P. Merkle, A. Smolic, K. Muller, and T. Wiegand, "Efficient prediction structures for multiview video coding," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no.11, Nov. 2007, pp. 1461–1473.

[67] W. Cai, G. Cheung, T. Kwon, and S.-J. Lee, "Optimized frame structure for interactive light field streaming with cooperative cache," in *IEEE International Conf. on Multimedia and Expo*, Barcelona, Spain, July 2011.

[68] G. Cheung, A. Ortega, and N.-M. Cheung, "Generation of redundant coding structure for interactive multiview streaming," in *Seventeenth International Packet Video Workshop*, Seattle, WA, May 2009.

[69] N.-M. Cheung, A. Ortega, and G. Cheung, "Distributed source coding techniques for interactive multiview video streaming," in *27th Picture Coding Symposium*, Chicago, IL, May 2009.

[70] J. Lee, I. Shin, and H. Park, "Adaptive intra-frame assignment and bit-rate estimation for variable gop length in h.264," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 10, pp. 1271 –1279, Oct. 2006.

[71] W. Gao and G. Cao, "On exploiting transient contact patterns for data forwarding in delay tolerant networks," in *Proceedings of the The 18th IEEE International Conference on Network Protocols*, ser. ICNP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 193–202.

[72] S. Ioannidis, A. Chaintreau, and L. Massoulie, "Optimal and scalable distribution of content updates over a mobile social network," in *INFOCOM 2009, IEEE*, April 2009, pp. 1422–1430.

[73] X. Wang, M. Chen, Z. Han, D. Wu, and T. Kwon, "Toss: Traffic offloading by social network service-based opportunistic sharing in mobile social networks," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 2346–2354.

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

[75] M. P. Karscig, *Principles of Economics*, 2nd ed. Harcourt., Inc., 2001.

[76] W. Cai, V. Leung, and L. Hu, "A cloudlet-assisted multiplayer cloud gaming system," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 144–152, 2014.

[77] 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.

[78] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware*, ser. Middleware'09, Berlin, Heidelberg, 2009, pp. 83–102.

[79] B. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, ser. MCS '10, New York, NY, USA, 2010, pp. 7:1–7:5.

[80] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11, New York, NY, USA, 2011, pp. 301–314.

[81] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.

[82] W. Cai and V. Leung, "Decomposed cloud games: Design principles and challenges," *Proceeding of The 2014 IEEE International Conference on Multimedia and Expo (ICME2014)*, July 2014.

[83] M. Mitchell, *An Introduction to Genetic Algorithms, 1998*. MIT Press, 1998.

[84] R. Poli and W. B. Langdon, "Schema theory for genetic programming with One-Point crossover and point mutation," *Evolutionary Computation*, vol. 6, no. 3, pp. 231–252, 1998.

[85] K. A. D. Jong and W. M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of Mathematics and Artificial Intelligence*, vol. 5, no. 1, pp. 1–26, Mar. 1992.

[86] P. Langley, J. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, vol. 10, no. 2, pp. 141 – 160, 2009.

[87] M. O'docherty, *Object-Oriented Analysis & Design*. John Wiley & Sons, 2005.

[88] W. Cai, X. Wang, M. Chen, and Y. Zhang, "Mmoprg traffic measurement, modeling and generator over wifi and wimax," in *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, Dec. 2010, pp. 1–5.

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

[90] D. Goldman, "Google unveils 'project glass' virtual-reality glasses," in *Money*, Apr. 2012.

Appendix A

Proof of Theorem 2.6

In the $n \times n$ solution matrix M, we define: *i) Two-node loop:* $\exists\ i, j \in \{1, 2, ..., n\}$ *and* $i \neq j$, *s.t.* $M_{ij} \cdot M_{ji} = 1$, *ii) Three-node loop:* $\exists\ i, j, k \in \{1, 2, ..., n\}$ *and* $i \neq j, j \neq k, k \neq i$, *s.t.* $M_{ij} \cdot M_{jk} \cdot M_{ki} = 1$, *iii) Similarly for higher order loops.* Denote $Q = M^2$, such that,

$$q_{ij} = \sum_{h=1}^{n} M_{ih} \cdot M_{hj} \tag{A.1}$$

Note that, since $M_{ij} \in \{0, 1\}, M_{ij}^k = M_{ij}$ for $k = 1, 2, ..., n$, then we derive:

$$\begin{aligned} q_{ii} &= \sum_{h=1}^{n} M_{ih} \cdot M_{hi} = M_{ii}^2 + \sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi} \\ &= M_{ii} + \sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi} \end{aligned} \tag{A.2}$$

$$\begin{aligned} tr(Q) &= \sum_{i=1}^{n} q_{ii} = \sum_{i=1}^{n} (M_{ii} + \sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi}) \\ &= \sum_{i=1}^{n} M_{ii} + \sum_{i=1}^{n} (\sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi}) \\ &= tr(M) + \sum_{i=1}^{n} (\sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi}) \end{aligned} \tag{A.3}$$

Since $M_{ij} \in \{0, 1\}$, if there is no *two-node loop*, then,

$$M_{ij} \cdot M_{ji} = 0, \forall i, j \in \{1, 2, ..., n\} \; and \; i \neq j \tag{A.4}$$

Thus,

$$\sum_{i=1}^{n} (\sum_{\substack{h=1 \\ h \neq i}}^{n} M_{ih} \cdot M_{hi}) = 0 \tag{A.5}$$

therefore,

$$tr(Q) = tr(M^2) = tr(M) \tag{A.6}$$

Now, let $X = M^3 = Q \cdot M$, if there is no two-node loop, we derive:

$$\begin{aligned} x_{ii} &= \sum_{h=1}^{n} q_{ih} \cdot M_{hi} \; = M_{ii} \cdot q_{ii} + \sum_{\substack{h=1 \\ h \neq i}}^{n} q_{ih} \cdot M_{hi} \\ &= M_{ii}^3 + \sum_{\substack{h=1 \\ h \neq i}}^{n} q_{ih} \cdot M_{hi} \; = M_{ii} + \sum_{\substack{h=1 \\ h \neq i}}^{n} q_{ih} \cdot M_{hi} \\ &= M_{ii} + \sum_{\substack{h=1 \\ h \neq i}}^{n} (\sum_{\substack{v=1 \\ v \neq i,h}}^{n} M_{iv} \cdot M_{vh} \cdot M_{hi}) \end{aligned} \tag{A.7}$$

Thus,

$$\begin{aligned} tr(X) &= \sum_{i=1}^{n} x_{ii} = \sum_{i=1}^{n}(M_{ii} + \sum_{\substack{h=1 \\ h\neq i}}^{n} \sum_{\substack{v=1 \\ v\neq i,h}}^{n} M_{iv} \cdot M_{vh} \cdot M_{hi}) \\ &= \sum_{i=1}^{n} M_{ii} + \sum_{i=1}^{n} \sum_{\substack{h=1 \\ h\neq i}}^{n} \sum_{\substack{v=1 \\ v\neq i,h}}^{n} M_{iv} \cdot M_{vh} \cdot M_{hi} \\ &= tr(M) + \sum_{i=1}^{n} \sum_{\substack{h=1 \\ h\neq i}}^{n} \sum_{\substack{v=1 \\ v\neq i,h}}^{n} M_{iv} \cdot M_{vh} \cdot M_{hi} \end{aligned} \tag{A.8}$$

Since $M_{ij} \in \{0, 1\}$, if there is no *three-node loop*,

$$\forall i, v, h \in \{1, 2, ..., n\}, i \neq v, v \neq h, h \neq i$$

$$M_{iv} \cdot M_{vh} \cdot M_{hi} = 0 \tag{A.9}$$

thus,

$$\sum_{i=1}^{n} \sum_{\substack{h=1 \\ h\neq i}}^{n} \sum_{\substack{v=1 \\ v\neq i,h}}^{n} M_{iv} \cdot M_{vh} \cdot M_{hi} = 0 \tag{A.10}$$

Therefore,

$$tr(X) = tr(M^3) = tr(M) \tag{A.11}$$

Using the same approach for higher order loops, we conclude that, to guarantee loop-free solution, the following equation shall be satisfied:

$$\forall p \in \{1, 2, ..., n\}, tr(M^p) = tr(M) \tag{A.12}$$