MCG Test-bed: An Experimental Test-bed for Mobile Cloud Gaming

Wei Cai
University of British Columbia
Vancouver, Canada
weicai@ece.ubc.ca

Conghui Zhou
We Software Limited
Hong Kong
neio.zhou@gmail.com

Minchen Li
Zhejiang University
Hangzhou, China
minchen@zju.edu.cn

Xiuhua Li
University of British Columbia
Vancouver, Canada
lixiuhua@ece.ubc.ca

Victor C.M. Leung
University of British Columbia
Vancouver, Canada
vleung@ece.ubc.ca

ABSTRACT

Conventional cloud gaming benefits from many aspects by executing the game engine in the cloud and streaming the gaming videos to players' terminals through network. However, this type of cloud gaming service can not guarantee stable Quality of Service (QoS) in mobile scenarios, subject to the diversity of end-user devices and frequent changes in network quality. To address this issue, a component-based cloud gaming platform for mobile devices has been proposed, where the decomposed game can be partially offloaded to the cloud, achieving load balancing between cloud and terminals. In this work, we implemented the very first experimental test-bed for component-based mobile cloud gaming. Three game prototypes are built on our test-bed, in order to demonstrate its feasibility and efficiency. Experiments have been conducted to show that intelligent partitioning leads to better system performance, such as lower response latency and higher frame rate.

Categories and Subject Descriptors

C.5.0 [**Computer System Implementation**]: [General]; D.2.10 [**Software Engineering**]: Design—*Methodologies*; K.8.0 [**Personal Computing**]: General—*Games*

Keywords

Cloud Gaming, Mobile Games, Test-bed

1. INTRODUCTION

Gaming industry is evolving itself to a new stage that aims to provide mobile gaming experience anywhere anytime to their customers. Researchers, developers and operators are eager for novel cross-platform solutions to video games, which is responsive to the variety of players' screen sizes, device hardware capacity, operating systems and interactional controllers. This demand has significantly soared, along with the arising number of mobile terminals in modern society. The cloud, well recognized as the next generation computing modality, has attracted attentions in video gaming service provisioning. Conventional cloud gaming solutions, including industrial systems (e.g. OnLive[1], Gaikai[2], G-Cluster[3], etc.) and research test-beds (e.g. GamingAnywhere[1]), exhibit a streaming-based approach, in which the video games are executed in cloud servers and the gaming videos are streamed to the players' terminals over network. In reverse, the players' inputs are transmitted to the cloud server to replicate the players' actions. [2]. The streaming-based cloud gaming platform brings many advantages, such as cross-platform distribution, anti-piracy, overcoming the hardware restriction of terminals, reducing the development and maintenance fee, etc.

Nevertheless, things go different when the scenario turns to 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 quality of service of the existing system can not be guaranteed, since real-time video transmission requires low-latency and high-bandwidth network access, while we are still suffered from the unstable quality of Internet infrastructure [3][4]. On the other hand, the capacity of mobile terminals has made remarkable enhancement, thanks to the rapid development of hardware design and technologies. Consequently, [5] has stated that there shall be alternatives toward gaming as a service (GaaS). Our previous work [6] has proposed a component-based cloud gaming framework, which consider game application as decomposed software to be dynamically allocated between cloud and terminals, according to terminals' computational power and network quality. In other words, the system will dynamically and partially offload the game to the cloud [7], so that the mobile terminal utilizes the rich resources from the cloud to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiGames'15, May 19, 2015, Florence, Italy.
Copyright © 2015 ACM 978-1-4503-3499-0/15/05 ...$15.00.
http://dx.doi.org/10.1145/2751496.2751501.

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

enhance its functionality. Thus, mobile cloud gaming becomes possible.

However, there is still no existing experimental environment for component-based cloud gaming platform. In this work, we develop the world's very first test-bed for component-based mobile cloud gaming, which learns about the status of terminalą́s environment and adapts the cloud gaming service to these evaluations. With the test-bed, we develop 3 game prototypes to demonstrate the efficiency of the proposed framework. The remainder of the paper is as follows. We review related work in Section 2 and briefly introduce the test-bed implementation in Section 3, respectively. Afterwards, we develop three game prototypes to conduct experiments in Section 4. Section 5 concludes the paper.

2. RELATED WORK

2.1 Cloud Gaming Architecture

Adaptive mobile video streaming architectures have been widely studied for mobile cloud games. The framework in [8] monitors the end-to-end network status and alters the video encoding parameters for cloud games accordingly. This work considers dynamic video encoding adaptation focusing on improving the user-perceived quality of a Gaming-on-Demand service, which is a highly demanding interactive multimedia application based on a client-server infrastructure and video streaming. Similarly, a recent work [4][9] proposes a Remote Server Based Mobile Gaming (RSBMG) architecture and develops a set of application layer optimization techniques to ensure acceptable gaming response time and video quality in the remote server based approach. The techniques include downlink gaming video rate adaptation, uplink delay optimization, and client play-out delay adaptation.

2.2 Partitioning Solution

To facilitate intelligent resource allocation, the cloud games should support dynamic partitioning between cloud and mobile terminal. There has been some work on partitioning of mobile applications. [10] first introduces a K-step algorithm as a dynamic solution where the partition is calculated on-the-fly, once a mobile connects and communicates its resources. Furthermore, according to [11], there is no single partitioning that fits all due to environment heterogeneity (device, network, and cloud) and workload. Consequently, they proposed a system that can seamlessly adapt to different environments and workloads by dynamically instantiating what partitioning to use between weak devices and clouds. An implementation [12] called CloneCloud is a flexible application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly off-load part of their execution from mobile devices onto device clones operating in a computational cloud. However, these works requires the application to be completely installed in both the mobile terminal and the virtual machine residing in the cloud.

3. TEST-BED IMPLMENTATION

3.1 Component-based Framework

As stated in our previous work [6], the proposed test-bed considers game software as inter-connected components, which function as cooperative modules via inter-component message transfer. In other words, a game application is decomposed into a number of pieces, which either executed in the cloud or the players' terminal, according to the status of devices and network quality. Consequently, the test-bed is required to be capable in perceiving the application runtime environment, making decisions of workload allocation, and facilitating the dynamic inter-connectivity between cloud/terminal components. Due to the limitation of paper length, we would not mention the detailed algorithms and mechanisms for the framework design, which has been described in [13].

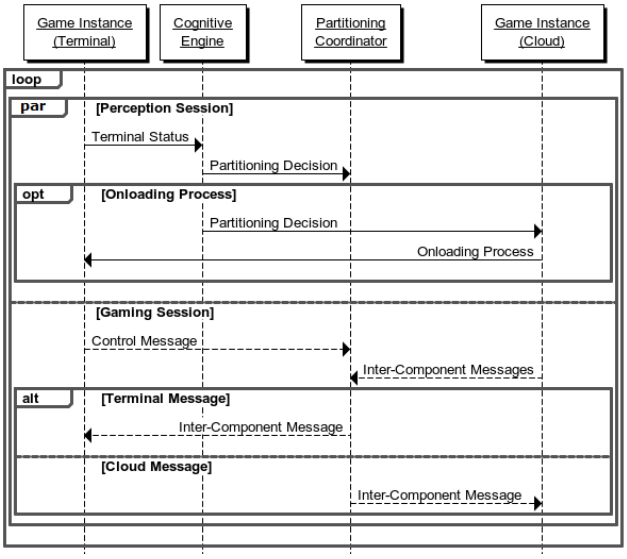


Figure 1: Sequence Diagram for Cognitive Engine of MCG Test-bed

To help readers of this paper understand the core workflow of MCG test-bed, we illustrate a sequence diagram for its cognitive engine in Fig. 1. To start the gaming, the cloud server first dispatches a JavaScript engine (including 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 lack of sufficient components to perform optimal partitioning.

3.2 Test-bed Development

To develop the MCG test-bed, a programming model that facilitates code migration and execution between cloud and terminals is in need. This requirement leads us to JavaScript, which exhibits as scripts in client-side and also powerful back-end language in Node.js[4] software system that de-

[4]http://nodejs.org/

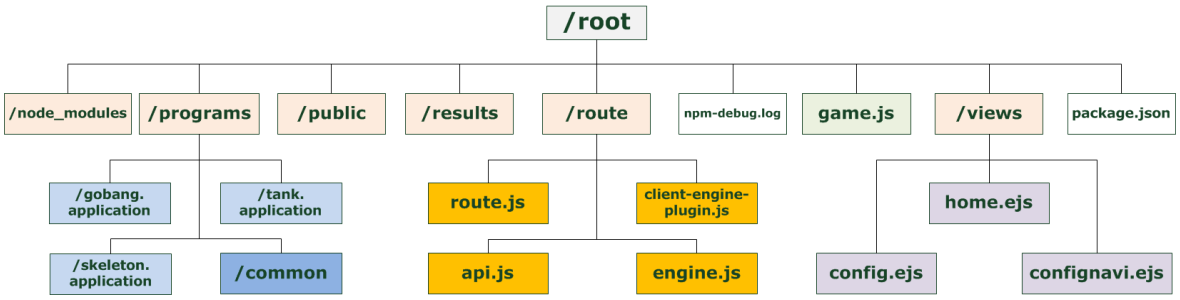


Figure 2: Deployment Directory of MCG Test-bed

signed for writing scalable Internet applications, notably web servers. For the mobile client, we embed a WebKit-based browser into our Android application to parse and execute the JavaScript mobile agent. Even though we currently only provide Android app, other mobile operating systems supporting browsers will work with MCG test-bed after a small portion of modification.

As depicted in Fig. 2, besides the standard elements of Node.js project (*package.json*, *npm-debug.log* and *node_modules* directory that imports dependent Node.js libaries), the MCG test-bed project is organized by Express[5], a Node.js web application framework following Model-View-Controller (MVC) software architectural pattern. Following is the deployment of our test-bed:

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

- */views*: as the portal of MCG test-bed, 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 test-bed, so that the players can click through the icons to access the game.

- */route*: as the test-bed'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 application programming interfaces (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[6], which is a set of Windows 8 template extended from Bootstrap[7] framework.

- */application*: for our application developers, their applications shall 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.

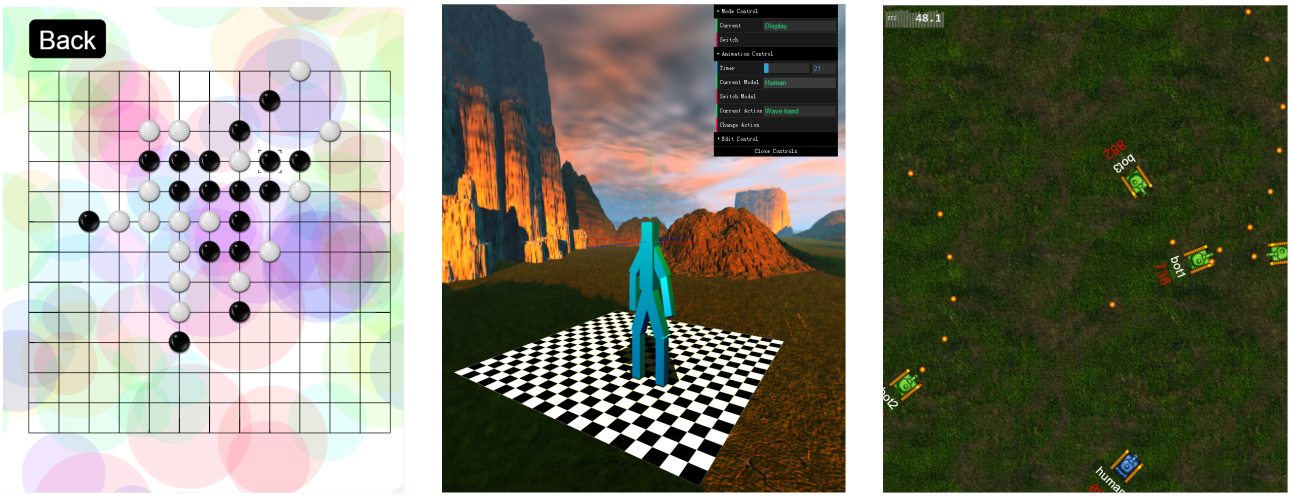
As a test-bed that supports component-based games, we provide a set of APIs to encapsulate lower layer partitioning for game developers. To acknowledge an inter-component invocation in MCG test-bed, the developers shall simply add a "$$" mark before the name of the components when they are invoked in the code (e.g. $$$componentX$({$msg : args$}); stands for an invocation of $componentX$ with a parameter of *args* passing as a message).

3.3 The Administration Center

Figure 3: Administration Center of Test-bed

Fig. 3 illustrates a screenshot of the MCG test-bed 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

[5]http://expressjs.com/
[6]http://metroui.org.ua/
[7]http://getbootstrap.com/



(a) Gobang Game (b) 3D Skeletal Game Engine (c) Robocode Tank Game

Figure 4: Screenshots of Game Prototypes on MCG Test-bed

graphical user interface. Note that, if the *AUTO OPTIMIZATION SWITCH* is turn 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. PROTOTYPE AND EXPERIMENTS

To verify the feasibility and efficiency of MCG tesbted, we develop and deploy three prototypes in this section, including a Gobang game, a 3-dimensional (3D) skeletal game engine, and a Robocode tank game. Runtime screenshots of the three prototypes are illustrated in Fig.4 and their experimental results are discussed as follows.

4.1 Gobang Game

Gobang game (also known as Gomoku or Five in a Row) is an abstract strategy board game that players alternate in placing chess of their color on an empty intersection of 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 MCG test-bed 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 on these two different environments.

Three types of devices are employed in our evaluation, including an ASUS windows 7 personal computer (PC) with Intel Pentium G630 @2.70GHz central processor unit (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 Android 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 MCG test-bed hosting in Amazon Elastic Compute Cloud (EC2)[8].

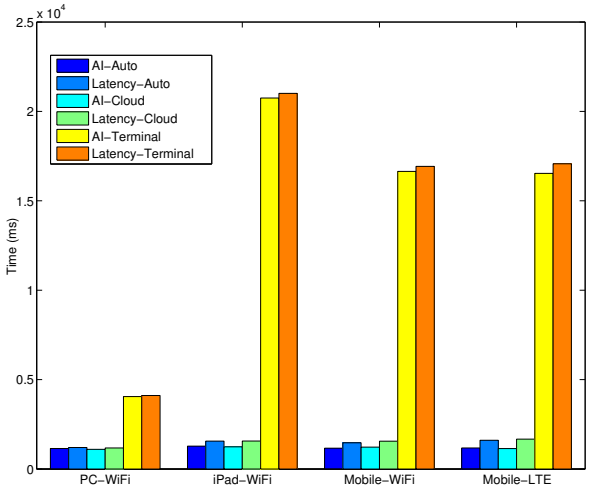


Figure 5: Response Latency Comparison in Gobang Game Prototype

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 (Test-bed 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 play-

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

er placing chess and the time the AI placing chess. These measurements are depicted as six schemes in Fig. 5.

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. It indicates 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.

4.2 3D Skeletal Game Engine

The 3D skeletal engine, our second prototype, aims to challenge MCG test-bed's capacity on rendering 3D game scenes. Besides the complexity of AI, modern games are also prone to create fantastic game scenes that consumes huge amount of terminal resources. Recent work [14] 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 known as 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. The developed 3D Skeletal Game Engine is consisted of a animation editor and a 3D rendering module, which computes and draws action animations for a human and a dog, respectively. The implementation screenshot of a four-component prototype is illustrated in Fig. 4(b).

To validate and demonstrate the MCG test-bed's feature of cognitive task allocation to the network quality, we perform experiments to measure the fluency of rendered animations by the numeric value of frame per second (FPS). In order to explicitly control the network 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[9] to control the bandwidth of NodeJS process in the cloud, for the purpose of simulating the variance of network quality in real-world cases. We design our experiments in two aspects. First, there shall be comparisons between automatic optimization and all potential partitioning schemes. Since the prototype contains four components, an iteration of possible partitioning makes 2^4 schemes. Therefore, we divide the total experiment time into equal 16 slides for each scheme. Second, we also concern about different system performance over different network bandwidth. Hence, we repeat the experiments three times, with bandwidth settings of 1000 KBps, 500 KBps and 100 KBps, respectively.

[9]https://netbalancer.com/

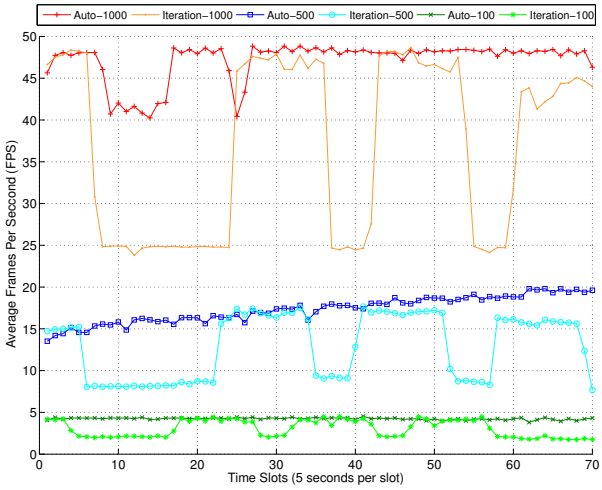


Figure 6: Game QoS (FPS) Comparison in 3D Skeletal Game Engine

Fig.6 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 KBPS to 500 KBPS. Things get worse if the network bandwidth is reduced to 100 KBPS, MCG test-bed can only render the 3D skeleton at FPS rate of 5, which is only 10% of the 1000KBPS case. The good part of this experiment is that, it proves the efficiency of MCG test-bed. 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 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.

4.3 Robocode Tank Game

The idea of third game prototype, Robocode Tank, comes from a famous open source educational game Robocode[10], 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 game prototype inherits all features of Robocode and places an additional tank controlled by players into the battlefield.

Since the Robocode tank system performance is determined by the varying complexities of tank AIs, here we only validate the cognitive capacity of MCG test-bed. To record FPS traces, three players were engaged in the EC2 hosted the tank game on previous mentioned LG G2 Android smartphone through Fido LTE network. Four AI-controlled tanks in the battlefield make strategy decision at 1 second interval. As shown in Fig. 7, these players all experienced

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

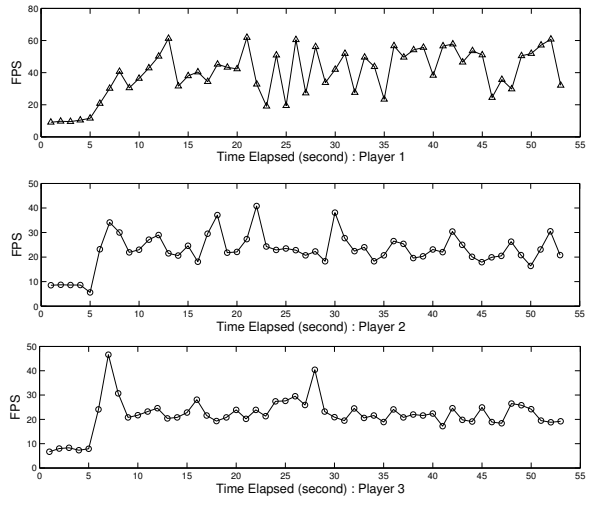


Figure 7: Game QoS (FPS) Enhancement for Robocode Tank Game

very low FPS rate at the beginning of the gaming sessions, while the test-bed eventually provided an optimal partitioning solutions for them. Note that, the solution search time for these three players are distinct from each other and optimal FPS is fluctuating, according to the everchanging network quality and game contents (the tank interactions in this game).

5. CONCLUSION AND FUTURE WORK

In this paper, we have described MCG, the world's first experimental test-bed for mobile cloud gaming. Unlike conventional research and developing ideas of cloud games, MCG test-bed facilitates component-based games with unique features of dynamic partitioning. We discussed the component-based framework and core workflow of MCG test-bed, while explain the selection of implementation technologies. Three game prototypes are developed and deployed to demonstrate the validation and efficiency of proposed test-bed. For future work, we focus on the following directions: i) toward better scheduling and resource management, we seek better measurement methods for components execution performance, both in cloud and mobile devices; ii) besides the latency-related optimizations, we are looking for more sophisticated mathematic models that consider larger perspectives of the system, such as computational efficiency, network bandwidth minimization, battery preservation, etc.

6. ACKNOWLEDGMENT

This work is supported by a University of British Columbia Four Year Doctoral Fellowship, funding from the Natural Sciences and Engineering Research Council, and UBC Work Learn International Undergraduate Research Award.

7. REFERENCES

[1] 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.*, 10(1s):10:1—-10:25, January 2014.

[2] 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*, pages 1 –7, 30 2009-dec. 4 2009.

[3] 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*, MM '11, pages 1269–1272, New York, NY, USA, 2011. ACM.

[4] 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)*, pages 1–6, USA, 2010.

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

[6] W. Cai, C. Zhou, V. Leung, and M. Chen. A cognitive platform for mobile cloud gaming. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013)*, 2013.

[7] K. Yang, S. Ou, and H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *Communications Magazine, IEEE*, 46(1):56–63, 2008.

[8] S. Jarvinen, J. 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*, volume 2, pages 994 – 997, jan. 2006.

[9] 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*, pages 1–6, 2010.

[10] 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*, Middleware'09, pages 83–102, Berlin, Heidelberg, 2009.

[11] 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*, MCS '10, pages 7:1–7:5, New York, NY, USA, 2010.

[12] 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*, EuroSys '11, pages 301–314, New York, NY, USA, 2011.

[13] W. Cai, M. Chen, C. Zhou, V. Leung, and H. Chan. Resource management for cognitive cloud gaming. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3456–3461, June 2014.

[14] 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*, pages 340–345, April 2014.