Balancing Cloud and Mobile Terminal: An Empirical Study on Decomposed Cloud Gaming

Wei Cai[1], Chaojie Zhu[1,2], Yuanfang Chi[1] and Victor C.M. Leung[1]
[1]Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada
[2]School of Electronics, Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
Email: {weicai, yuanchi, vleung}@ece.ubc.ca, zackszcj@sjtu.edu.cn

***Abstract*—Decomposed cloud gaming is a novel paradigm to deliver gaming as a service to mobile terminals. In such a system, video games are decomposed into software components that are cognitive to players' behavior and execution environments in both cloud and mobile devices. However, in order to balance the workload distribution between cloud and mobile terminals in a real system, there are still a number of questions to be answered, which requires empirical studies with detailed measurement and analysis. In this work, we identify these questions and answer them by measuring the execution and networking performance data from a tank game prototype in our test-bed. Based on our observations on the affecting factors on the system performance, we design and implement the cognitive capacity for the system to dynamically adapt its service to different terminals and networks.**

I. Introduction

Cloud gaming [1] refers to novel gaming service that utilizes cloud computing to facilitate better system performance. In conventional cloud gaming system, video games are hosted in cloud servers and their gaming videos are delivered via the Internet to all kind of terminals, such as desktop PCs, smartphones, etc. In reverse, the players' inputs are delivered to a cloud server and accepted by the game content server directly [2]. Hosting games in the cloud introduce benefits to system maintenance and service provisioning, due to its cross-platform feature. Realizing its bright future, the industry has been leading the way. Even though G-cluster[1], one of the earliest cloud gaming providers, has filed for bankruptcy on May 2016, Sony PlayStation Now[2] is still in operation. It serves as a cloud gaming service from Sony Interactive Entertainment, after absorbing two cloud gaming forerunners, OnLive and Gaikai. As an interactive real-time video streaming system, the bottleneck of cloud gaming goes to unpredictable latency and insufficient bandwidth of Internet connectivity, especially in the scenario of mobile networks. Under this circumstance, conventional real-time video compression techniques and adaptive transmission [3] [4] [5] are introduced to further optimize gamers' quality of experience (QoE). The foundation of these studies is based on a common assumption: gamer players would prefer to scarify video quality to gain smoother playing experience in insufficient network QoS supplement.

However, the state-of-art implementations are still far away from complete satisfactory: it is evident that cloud gaming is still not accepted by the mass market. To this end, another direction of research [6] brings software decomposition into cloud gaming area. Their work decomposes the rendering module of a game into two sub-modules: one executing in the cloud to render the scenes and create the rendering instructions for game scene updates, while the other interprets the rendering instructions and transmits them to the mobile devices for local execution. By decoupling the creation of rendering instructions from its execution and transmitting only small-sized rendering instructions over the Internet, the communication burden caused by video transmissions is eased, hence meeting the challenges caused by the limitations of the mobile networks. Motivated by this decomposition idea, we proposed the decomposed cloud gaming paradigm [7], where a game program is decomposed into inter-dependent components that can be flexibly distributed to either the cloud or terminal for execution. Thus, player's QoE can be cognitively optimized by dynamically allocating components according to various system parameters, including different terminal capacities (mobile or stationary devices), heterogeneous networks, and distinct players' behaviors. We demonstrated the design and implementation of our proposal with a decomposed cloud gaming test-bed and three prototype games [8]. However, to cognitively balance the workload between cloud and mobile terminals, we still a number of critical engineering issues to be addressed. These unanswered questions include:

1) Will the concurrency of multiple components affect the computing performance of cloud and terminals?
2) Will the intensive computational workload affect the computing performance of cloud and terminals?
3) Will hardware specification affect the computing performance of cloud and terminals?
4) Will the message exchange between components within cloud affect the system performance?
5) Will the message exchange between components within terminal affect the system performance?
6) Will the message networking between cloud and terminal affect the system performance? What can the performance alter our partitioning strategy?
7) Will decomposed cloud gaming enhance players' gaming experience, or it's just a gaudy concept?

In this paper, we answer above questions by conducting an empirical study on component execution performance of a

[1]http://www.gcluster.com/
[2]https://www.playstation.com/en-ca/explore/playstationnow/

tank game prototype. Based on our observations, we design and implement the cognitive engine to optimize the workload balancing between cloud and terminals and validate its effectiveness by experiments.

The remainder of the paper is as follows. We review related work in Section II and briefly present the system implementation for decomposed cloud gaming in Section III, respectively. Afterwards, we perform measurements in Section IV to study the execution performance of components in different environments. Based on the observation, we propose the cognitive engine implementation in Section V and evaluate our algorithm in VI. Section VII concludes the paper.

II. Related Work

A. Cloud Gaming

Cloud gaming platforms can be categorized into two classes: *transparent platforms* and *non-transparent platforms*. Traditional cloud gaming services, such as PlayStation Now, belong to the transparent platforms [9], which run unmodified games at the expense of potentially suboptimal performance. In contrast, the non-transparent platforms [10][11] require augmenting and recompiling existing games to leverage unique features for better gaming experience, which may potentially be time-consuming, expensive, and error-prone. In this work, our proposed decomposed cloud gaming system falls into the later category.

B. Quality of Experience in Cloud Gaming

Maintaining an acceptable quality of experience (QoE) for the game players is an imperative design concern for cloud gaming systems. To provide cloud gaming service, the relationships between cloud gaming QoE and QoS are different for different implementation architectures. For traditional cloud gaming, many subjective user studies have been conducted to demonstrate the relationships between cloud gaming QoE and QoS, including game genres, video encoding factors, central processing unit (CPU) load, memory usage, link bandwidth utilization, response latency and the game's real-time strictness [12], category of gaming scenes, and network characteristics (bit rates, packet sizes, and inter-packet times) [13]. Nevertheless, with the proposed cognitive cloud gaming platform, the QoE to QoS mapping needs to be redefined due to the adaptive nature of the platform. There is relatively little research done in this respect. In current work, we consider the fluency of game execution the most important factor that affects the players' QoE. Therefore, we focus on optimization of game rendering, which specifically represented by the value of frame per second (FPS).

C. 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 the partitioning of mobile applications. [14] 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 [15], 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 [16] called CloneCloud is a flexible application partitioner and execution run-time 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 require the application to be completely installed in both the mobile terminal and the virtual machine residing in the cloud.

III. System Implementation

A. Platform Overview

The decomposed cloud gaming platform considers game software as inter-connected components, which function as cooperative modules via inter-component message exchange. 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. 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* selects destination components, locally or remotely, to achieve dynamic resource allocation. This intelligent selection is performed by *Cognitive Decision Engine*. It requests information from the *Performance Prober*, which periodically reports its results in collecting data from *Execution Monitor*s in both cloud and terminal side. For further details about test-bed implementations, please refer to our previous publication [8].

B. Prototype Development and selection

To verify the feasibility and efficiency of the implemented platform, we developed a number of decomposed game prototypes, including 3D Skeleton Prototype, Gobang Game and Robocode Tank, as described in our previous work [17]. In this paper, our target is to design quantitative measurements and experiments to demonstrate the proposed features. Critical issues in designing such experiments include 1) how to select representative game prototypes; 2) how to measure the impact of computational capacities in the cloud and terminals; 3) how to measure the impact of network parameters. To this end, we select the Robocode Tank game prototype to study the execution and network status for components. We put a different quantity of tanks with distinct computational costs (represents the intensive computing for tanks' artificial intelligence) into the battlefield, in order to simulate a variety of gaming scenarios.

IV. Measurements

In this section, we perform measurements for computational capacity and the communication capacity. Our purpose is to design figure out how different systematic factors affect these two capacities. The factors we considered include the CPU and the memory of the cloud and the terminal, and the network between the cloud and the terminal.

A. Experimental Settings

To simulate the cloud, network and terminal environment, we set up an experimental test-bed in The University of British Columbia, Vancouver. We employed two personal computers (PCs) to simulate the environment of cloud and terminal. Both of these two PCs are equipped with 8 core CPU, 8 Gigabytes (GB) memory. During the experiment, we use BIOS settings to adjust the CPU cores and memory size, in order to simulate different hardware scenarios. Nodejs v4.4.7 is installed as the cloud server software, while Firefox 45.0.2 is adopted as the default client in the terminal. To connect cloud and terminal, we wired them to a Linksys Wireless Router WRT120N router so that they are inter-connected within one local area network (LAN). The Robocode tank game is deployed on port 8080 of cloud PC and the terminal accesses the game through cloud's local Internet protocol (IP) address. In order to control the network parameters between cloud and terminal, we installed NetLimiter 4.

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

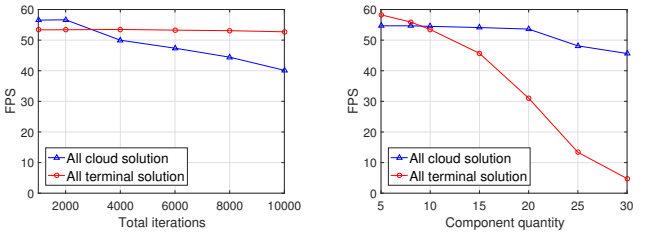
B. Computational Capacity

Our hypothesis is that the parameters affecting computational capacity include computing intensity and process concurrency. The former value indicates the computational complexity of a particular component, while the latter value indicates the ability of parallel execution for multiple components. In this work, we use the iterative executions of a segment of code to demonstrate the computing intensity, and the quantity of simultaneous components to simulate the process concurrency.

Before quantitatively study the influences of different factors, we conducted two case studies to discover the relationships between FPS and these two parameters. The specification of the cloud servers and the terminal in the case studies are listed in Table I.

TABLE I: Specification for Case Studies

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



(a) FPS and Iterations (b) FPS and Component Quantity

Fig. 1: The different computational capacities of the cloud and the terminal

The first case study sets 5 components running at a time with the same iteration and gradually increases the total iteration from 1000 times to 100000 times, maintaining all other factors unchanged. As illustrated in Fig. 1(a)), the cloud originally has a higher FPS when the total iteration is 1000 times, then slowly goes down, and get lower after the iteration increased to 4000 times. Comparatively, the FPS when all components run on the terminal is much more stable and keeps at a high level. The second case study restricts all components' total iteration to be 5000 times and gradually adds the number of components from 5 components to 30 components, maintaining all other factors unchanged. As shown in Fig. 1(b), the FPS when all components run on the terminal is higher originally, while it declines dramatically with the increment of the component quantity. The average FPS even gets lower than 10 when the component quantity comes to 30. In contrast, the curve of the cloud only declines about 10% when the component quantity gets larger than 20. Fig. 1 illustrates that the terminal can maintain a higher computation capacity with large iterations, while the cloud servers obviously can handle concurrency better. So when partitioning the components, these two parameters need to be taken into account to place the components to more suitable sides.

Inspired by these case studies, we designed a number of experiments to further validate the relationship between hardware specification and execution performance. Our results reveal that, CPU core quantity and memory size are considered the most important factors in computational performance. Therefore, we selected different combination of CPU core quantity and memory size, in order to derive the FPS values of game execution, which indicates the performance.

1) Iterations: To find out how CPU and memory may affect the cloud servers' and the terminal's behavior handling increasing iterations, we design experiments for all components executing in the cloud (Experiment C1 to C5) and

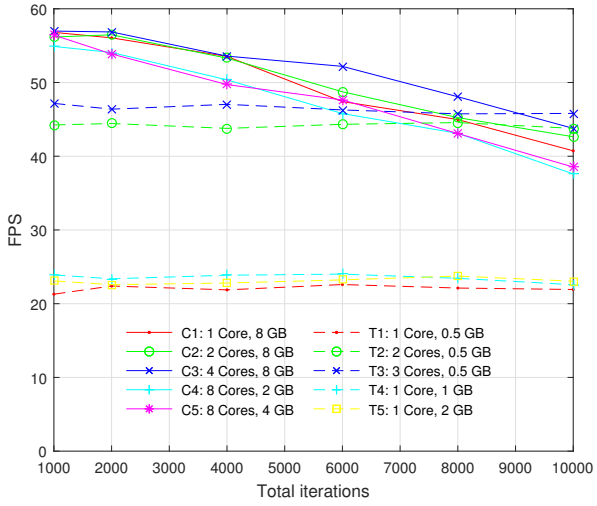


Fig. 2: The FPS under different experiments for the cloud

all components executing in the terminal (Experiment T1 to T5), with specifications shown in the legend of Fig. 2. We recorded the average FPS with each specification and illustrated the characteristic of the cloud and the terminal by Fig. 2. Apparently, the increment of either CPU cores or memory size will enhance the performance. Relatively speaking, memory size appears to be a more critical element in capacity improvement, since the average FPS values in C3 is higher than that of Experiments C4 and C5. On the terminal side, it turns out that the quantity of CPU core is the key point of its capacity instead. The FPS performances in T2 and T3 are significantly better than all of the others. In contrast, similar FPS results in other experiments show that increasing memory size does little to improve the computational capacity of the terminal.

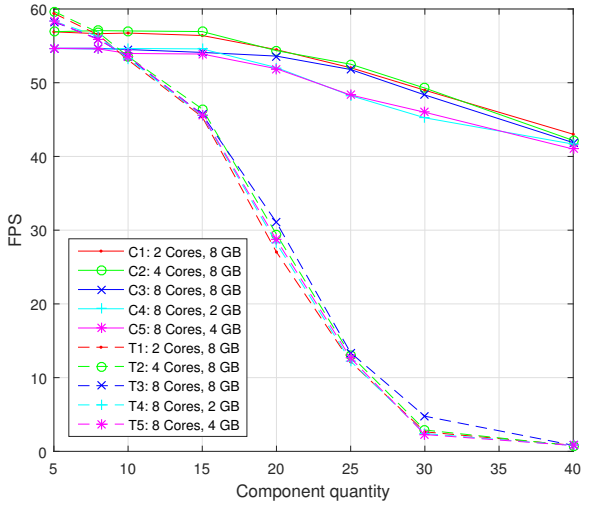


Fig. 3: The FPS under different experiments for the terminal

2) Concurrency: As mentioned in Section IV-B, concurrency, which indicates the ability of parallel execution for multiple components, is another important parameter affecting the computational capacity. The same to the iteration studies, we again set different combinations of specifications in cloud and terminal and record their FPS results as shown in Fig. 3. As predicted, more components executing at the same time will reduce the FPS value. However, unlike what we've found about iterations, there is no significant FPS difference among distinct hardware settings. This phenomenon implies that neither increasing CPU cores nor enlarging memory size will help improving the concurrency capacity in the terminal. In other words, improvements on hardware helps little in solving the concurrency issue. Comparing terminal and cloud, the rapid performance degradation of multiple components on terminal is an critical issue to be addressed.

C. Communication Capacity

Besides the computational capacity, the communication capacity between components is also critical in determining the system performance. We categorize the communications into three kinds: intra-cloud communications, intra-terminal communications and cloud-terminal communications. Also, we also characterize the communication with two parameters: the message length and the communication frequency. In our measurement, the message length is evaluated by bpm (bytes per message). In addition, we consider 20 movement steps of a tank as a batch and the frequency is evaluated by tpb (the times of communication per batch). Apparently, higher bmp value represents larger message payload in one transmission, while higher tpb value represents higher communication frequency. Default experiment settings are listed in Table II.

TABLE II: The specification for the communication capacity

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

We start our measurement from a simplified circular communication model: each component sends a message with the specified length to its next component at the specified frequency, such as Component No.1 sends a message to Component No.2 and the last component sends a message to Component No.1. Besides, in order to simulate a real scenario, we create following iterations for each component in a specific component group as listed in Table III.

TABLE III: The components for the communication capacity

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

1) Intra-Cloud Communications: indicates the capacity of transmitting data from one component to another within the cloud. We reveal how message length and frequency may affect the communication capacity by the different combinations. The FPS results under different communication settings are

shown in Table IV. The first row is considered as a baseline of the FPS. In other situations, the FPS results decline when messages get larger or more frequently sent. On the other hand, except the extreme situations where messages are as long as 262144 bytes, the performance is generally acceptable.

TABLE IV: The FPS of Intra-Cloud Communications

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

2) Intra-Terminal Communications: indicates the capacity of transmitting data from one component to another within the terminal. Owing to the fact that the performance is too bad if we still keep the experiments with 12 components, we removed the last six components in the experiments conducted in this section. The combinations of message length and frequency are altered accordingly due to the same reason. The FPS results in Table V is surprisingly low even the communication is reduced by our experiment settings. Relatively speaking, the communication frequency seems to be a more important factor. It is explained by the fact that the FPS is still above 30 when the 128-byte message is sent for 60 times per batch while the FPS is lower than 5 when the frequency comes to be 80 tpb and the message length is unchanged. The result in this table shows us that the communication on the terminal side is a huge burden that should be avoided.

TABLE V: The FPS of Intra-Terminal Communications

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

3) Cloud-Terminal Communications: indicates the capacity of transmitting data from one component to another between the cloud and the terminal. As mentioned before, every component sends a message to the component next to it in a circular model, such as Component#1 sends all messages to Component#2, and etc. We placed the components according to its component number so that all communications can be guaranteed to be cloud-terminal ones. The specific side information of components is shown in Table VI.

TABLE VI: The partition of the Components

	Cloud side	Terminal side
Component No.	1, 3, 5, 7, 9, 11	2, 4, 6, 8, 10, 12

The result in Table VII turns out that the capacity of cloud-terminal communication is a moderate one. When the packets are small and not so frequently sent, the FPS is high and stable, while when it comes to those large messages with small interval the FPS drops significantly like what the intra-terminal communication does. Hence, during the partitioning, we can still deliver the components with high iterations, considering both computational and communication capacity.

TABLE VII: The FPS of Cloud-Terminal Communications

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

However, above experiments were conducted with high-speed LAN, which implies that we have not considered the effect of network bandwidth. In order to better demonstrate the performance in real scenarios, we adopt 60 bpm and 128 tpb to conduct experiments under different restricted network bandwidths, which simulate the cases of Wi-Fi, 4G and 3G. Since the communications are designed to be synchronous operations in our tank game prototype, the bandwidth is a decisive factor in the communication performance. It can be explained by the dramatic declination of FPS in Table VIII.

TABLE VIII: The FPS of different network conditions

	3G	4G	Wi-Fi
Down limitation	93.75KB/s	0.5MB/s	3.75MB/s
Up limitation	31.25KB/s	0.375MB/s	1.875MB/s
FPS	0.006	5.98	41.82

D. Observation Summary

These measurements result in several conclusions regarding computational capacity and the communication capacity: 1) The cloud outperforms terminal in terms of component concurrency, while the terminal outperforms the cloud in terms of iterations, which implies higher computational capacity; 2) Increasing the memory capacity can be helpful for the cloud to improve its computational capacity, while adding more CPU cores is even more effective for the terminal; 3) The cost of intra-cloud communications is negligible, since it won't cause many declinations of the performance in most cases; 4) The intra-terminal communication drops the performance dramatically. Relatively, the frequency of the communication is a more important factor than the message length; 5) The impact of cloud-terminal communication on performance is between that of intra-cloud communication and that of intra-terminal communication. Communication frequency is an important factor, since it determines the number of round-trip times.

V. Cognitive Engine

With above observations, the optimal partitioning problem can be intrinsically transformed into the problem of seeking an optimal balance for the trade-off between cloud and terminals. Our target is to improve the performance on-the-fly

TABLE IX: Table of components in different experiments

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

and eventually obtain the best partition for the application in dynamic context. The partitioning algorithm we implemented in this work is a greedy algorithm. It moves the component with most iteration from the cloud to the terminal step by step until the FPS arrives its peak. We use a greedy algorithm because it is simple and fast, which guarantees the real-time players' QoE optimization. In our implementation of the cognitive engine, we divided the whole optimizing procedure into two operations: performance probe operation and partitioning operation.

The performance probe operation collects two kinds data needed for optimization. One is the real-time FPS of the Robocode tank game. The FPS values are derived from the terminal, so the terminal needs to send these data to the cloud. Another one is the execution time of each component. In a real scenario, to derive the number of iterations requires code and parameter analysis, which is described in CloneCloud [16]. In order to overcome this issue, we used a component's execution time as a rough estimation of its iteration. These data are collected by mobile agent [18].

The partitioning operation aims to find the optimal partitioning schemes under dynamic circumstances. It takes an attempt-confirm strategy or an attempt-reject strategy by comparing current FPS to previous values. During cognitive partitioning, the engine makes attempts of partitions and decides whether confirm or reject it according to the performance probe operation. After this procedure, the overall performance of the application will be optimized to satisfy players' QoE requirements. The pseudo-code of this algorithm is listed in Algorithm 1.

Algorithm 1 Greedy Partitioning Algorithm

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

According to the algorithm, the cognitive engine initializes a score for each partitioning scheme. The score is designed to decide whether the optimizing operation should be terminated. Once the score of one partition goes above one certain threshold (set to 3 in our experiments), the optimization will stop to provide a stable experience for the players. Once the gaming session starts, the system identifies the component that consumes the longest time and attempts to move it to the terminal. If the last attempt improves the performance, the cognitive engine confirms this attempt as a better partitioning. Otherwise, the system will reject the attempt. There are many reasons for FPS reduction, from the high component quantity on the terminal to the appearance of intra-terminal communication. No matter what the reason it is, the previous partition is apparently a better one. As a result, the cognitive engine adds the score of the previous partition and rolls the attempt back to the previous state.

VI. Experiments

In this section, we validate the effectiveness of the proposed cognitive engine. We designed experiments to simulate the practical cloud gaming scenarios. By applying the cognitive engine to these four experiments, the result shows whether the cognitive can do the optimization work and how much the performance is enhanced.

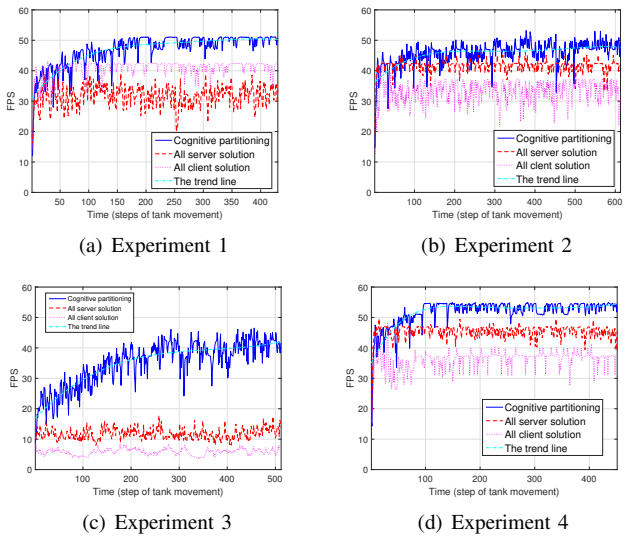
A. Experimental Settings

We realize that there are generally four kinds of situations that are common in game development: the simple games with fewer components, the complex games with a larger number of components, the games with only several high-iteration components leaving others low-iteration ones and the games mainly composed by high-iteration components. As a result, the experiments are in correspondence with these four situations by an elaborately designed combination of iterations and component quantity to make our simulation of the cloud gaming system to be more practical. The settings and purposes of the four experiments are listed in Table IX. With these settings, we conducted two experiments, one with no communication between components and one with networking with its next component, so that we can analyze the two factors separately.

B. Experimental Results

We start Robocode tank gaming sessions with no communications between components and record the FPS values along the time. We compare our proposed dynamic partitioning with two static approaches: all cloud solution and all terminal solutions, where the former scheme allocates all components in the cloud and the latter one assigns all components to the terminals. As illustrated in Fig. 4, the cognitive partitioning outperforms other solutions in terms of FPS in all four experiments. Especially in Experiment 3, we are able to enhance the

performance by 300%. This implies the efficiency of dynamic workload balancing.



(a) Experiment 1 (b) Experiment 2

(c) Experiment 3 (d) Experiment 4

Fig. 4: Simulations of the partition algorithm

The optimized partition of the components in different experiments is listed in Table X .

TABLE X: Partition Result without Communications

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

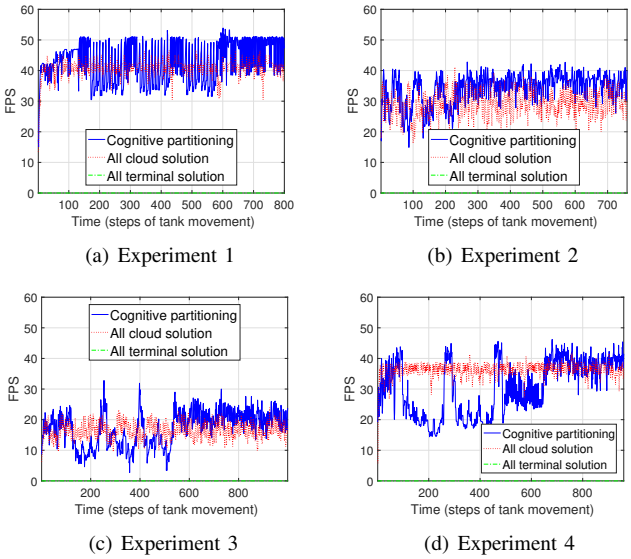
These experiments results prove two of our hypotheses: i) A component's execution time can be used to evaluate its quantity iteration. In all of these four experiments, the components being moved to the terminal are those components with most iterations, which meets our expectation. ii) In contrast to the cloud, terminals have better computational capacity when handling high-iteration components and worse computational capacity when handling concurrency is proved by the partitions. Experiment 4 contains more high-iteration components than Experiment 2, which leads to more terminal components as well.

Communication between components is also important in some game genres. For examples, some tanks might share their information with their teammates in order to perform cooperative battle. In this, work, we simulate these communications by transferring some redundant data between tanks. Parameters for communications are as listed in Table XI.

TABLE XI: Communication Information

No.	Frequency (tpb)	Message Length (bpm)
1	4	128
2	20	65536
3	100	256
4	80	4096

The FPS values during optimizing procedure are illustrated in Fig. 5 and the optimized partition is listed in Table XII.



(a) Experiment 1 (b) Experiment 2

(c) Experiment 3 (d) Experiment 4

Fig. 5: Simulations of the partition algorithm

TABLE XII: Partition Result with Communications

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

According to Fig. 5, the FPS improvement is not as significant as the ones in Fig. 4, because of the networking overhead in the remote component invocation. Also, the optimal solution discovery consumes more time, as there are more rejected attempts in the cognitive algorithm. However, the optimized partition still has higher FPS ahead of the situation when running all components on the cloud. Furthermore, in all of the experiments, the tank games cannot run when all components are placed on the terminal, which means the cloud gaming system is essential to run these kinds of games.

In addition, comparing the results in Table XII and Table X, we can find that even two communicating components with very high iterations are placed on the terminal side in the previous experiment without communication, such as the first two components in Experiment No.4, they are forced to be partitioned to different side due to the poor performance of intra-terminal communication. There seems to be an exception in Experiment No.1 where the 7th and 8th components are on the terminal though they have communication between them. However, that's because the messages are short and are sent infrequently. This phenomenon shows that the computational capacity and the communication capacity should be comprehensively considered and our cognitive engine is able to discern those different situations.

VII. Conclusion

As a novel paradigm, decomposed cloud gaming for mobile terminals is still under development. Engineering issues in decomposing game programs are still most critical challenges in this topic. In this paper, we seek solutions to these issues from an empirical approach. We prototyped a tank game application on our test-bed for us to measure the execution performance in detail. These measurements result in several conclusions to answer six questions from Section I regarding computational capacity and the communication capacity.

1) The cloud has a higher computational capacity about concurrency, which means the application can perform better when there is a great number of components running on the cloud than those running on the terminal.
2) The terminal has a higher computational capacity about iteration, which means the application can perform better when the components with high iterations running on the terminal than those running on the cloud.
3) Increasing the memory capacity helps the cloud to improve its computational capacity, while adding more CPU cores is even more effective for the terminal.
4) The cost of intra-cloud communications is negligible, since it won't cause many declinations of the performance in most cases.
5) The intra-terminal communication is a huge burden for the performance, which drops the performance dramatically. Relatively, the frequency of the communication is a more important factor than the message length. As a result, the cognitive engine should try to avoid the appearance of such frequently conducted communication.
6) The capacity of cloud-terminal communication is between that of intra-cloud communication and that of intra-terminal communication. It's more acceptable when the frequency is not big. So, moving some components without intra-terminal communication to run on the terminal is essential when its iteration is too high for the cloud to handle efficiently.

Based on our observation from measurement results, we designed and implemented a greedy cognitive engine for the test-bed. The purpose of the greedy algorithm is to push more components with the highest iterations to the terminal, in order to better utilize the terminals' high performance computing power on iterations. However, the cognitive nature will also avoid the cases that too many components running on the terminal, which may bring the concurrency issue and the intra-terminal communication issue. Our system will eventually reach the balance for this pair of trade-off. Experiments are conducted to evaluate and validate the engine. According to the results, the conclusions made by the measurement are proved to be valid and the cognitive engine we developed in this paper to be effective, which can guarantee a high QoE for the players. These experimental results are solid evidence to answered question 7 asked in Section I.

7) Cloud is not only an enhancement supplement but also an essential requirement for some gaming systems, as Fig. 5 demonstrated that some games won't be able to run without the support of cloud.

References

[1] W. Cai, R. Shea, C. Y. Huang, K. T. Chen, J. Liu, V. Leung, and C. H. Hsu, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2016.

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

[3] 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, pp. 994 – 997, jan. 2006.

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

[5] M. Hemmati, A. Javadtalab, A. Shirehjini, S. Shirmohammadi, and T. Arici, "Game as video: Bit rate reduction through adaptive object encoding," in *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'13)*, (Oslo, Norway), pp. 7–12, February 2013.

[6] S. Gorlatch, F. Meilaender, D. andGlinka, W. Zhang, and X. Liao, "Bringing mobile online games to clouds," in *Proceeding of 33rd IEEE International Conference on Computer Communications*, INFOCOM2014, 2014.

[7] W. Cai, H. C. B. Chan, X. Wang, and V. C. M. Leung, "Cognitive resource optimization for the decomposed cloud gaming platform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, pp. 2038–2051, Dec 2015.

[8] W. Cai, C. Zhou, M. Li, X. Li, and V. C. Leung, "Mcg test-bed: An experimental test-bed for mobile cloud gaming," in *Proceedings of the 2Nd Workshop on Mobile Gaming*, MobiGames '15, (New York, NY, USA), pp. 25–30, ACM, 2015.

[9] C. Huang, C. Hsu, Y. Chang, and K. Chen, "Gaminganywhere: an open cloud gaming system," in *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, (New York, NY, USA), pp. 36–47, 2013.

[10] 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*, MM '11, (New York, NY, USA), pp. 103–112, ACM, 2011.

[11] 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*, pp. 1–6, jul 2014.

[12] 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.

[13] 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*, pp. 1 –7, 30 2009-dec. 4 2009.

[14] 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, (Berlin, Heidelberg), pp. 83–102, 2009.

[15] 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, (New York, NY, USA), pp. 7:1–7:5, 2010.

[16] B.-G. 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, (New York, NY, USA), pp. 301–314, 2011.

[17] W. Cai, Z. Hong, X. Wang, H. C. B. Chan, and V. C. M. Leung, "Quality-of-experience optimization for a cloud gaming system with ad hoc cloudlet assistance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, pp. 2092–2104, Dec 2015.

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