DECOMPOSED CLOUD GAMES: DESIGN PRINCIPLES AND CHALLENGES

Wei Cai and Victor C.M. Leung

The University of British Columbia, Canada
{weicai,vleung}@ece.ubc.ca

ABSTRACT

To support cognitive resource allocation between cloud and terminals, the cloud gaming program is segmented into a set of collaborative components that are migratable from cloud to players' terminals. In contrary to conventional software architecture, its distributed execution manner introduces new challenges in system design and implementation. In this paper, we present our principle of constructing the decomposed cloud games and discuss the research issues from the perspectives of decomposition granularity, performance evaluation and response delay.

***Index Terms*—** cloud, game, decomposition, software

1. INTRODUCTION

Providing Gaming as a Service (GaaS) has been exhibited several advantages, including scalability, cost-effeciency, ubiquitous and multiple-platform support, effective anti-piracy solution, and click-and-play [1]. Therefore, not only the academia, but also the industry raise great expectations on the development of GaaS solutions. Traditional cloud gaming approaches adopts gaming-on-demand, also known as Remote Rendering GaaS (RR-GaaS) model, to host video games in cloud servers and transmit the gaming video frames to players' terminals over the Internet. However, the contradiction between huge bandwidth consumption of real-time video consumption and limited network bandwidth is still an open issue, especially when the terminals are accessing the cloud gaming services through mobile networks [2]. To solve this problem, a Local Rendering GaaS (LR-GaaS) model (i.e. browser games [3]) attempts to delegate the game video rendering engine to the terminals, thus, fundamentally eliminates the vide high network burden of real-time video transmission.

Intuitively, the fundamental difference between RR-GaaS and LR-GaaS is the allocation of rendering engine. In fact, if we further investigate the software architecture, a video game application is essentially a loop procedure that enables the interaction between players and game logics. This particular procedure might contain specified input/output (I/O) methods and involve information exchange between multiple players. Nevertheless, similar to most of the software systems, a game program is composed a set of software building blocks with distinct functionalities, which interacts with each other to provide gaming contents that receive players' instructions and response to them. We denoted these building blocks as components. This observation gives rise to the following questions: given the components are resource consumers either executed in cloud or players' terminals, is there a component allocation scheme that maximized the efficiency of the cloud gaming system? Is there a more flexible solution adapting the cloud gaming service to the varying circumstance, e.g. the unstable quality of network connectivity?

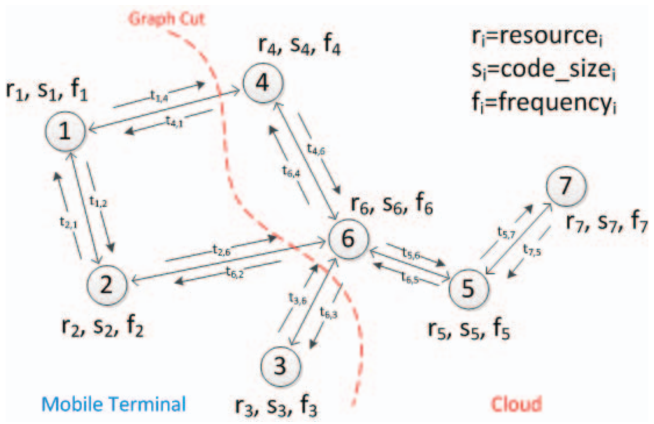


Fig. 1. Components Allocation for Decomposed Games

To answer these questions, the work [4] proposes a new design methodology: a decomposed cloud game is hosted by a cognitive platform in the cloud, which analyzes system context and intelligently determines the set of inter-dependant components' execution environment: cloud server or players terminal. Accordingly, the platform dispatches components to the terminals as stringified execution code and supports both local and remote interactions between components. Theoretically, it seeks an optimal cut in a graph that cognitively adopts to environment. See Fig. 1 for an illustration.

In this paper, we discuss the design principles and challenges of decomposed cloud games. The outline of the paper is as follows. We first review related work in Section 2 and then study the decomposition granularity and their challenges in Section 3. Afterwards, we investigate the perfor-

Thanks to XYZ agency for funding.

mance evaluation methods in Section 4. An analysis of response delay for decomposed cloud game is presented in Section 5. Concluding remarks is presented in Section 6 .

2. RELATED WORK

Resource allocation optimization problem is intrinsically a group of dynamic partitioning problem for multiple devices. Researches on the dynamic partitioning between cloud and users' mobile terminal have been conducted for general-purpose applications. The work [5] first introduces a K-step algorithm to compute partitioning on-the-fly, when a phone connects to the server and specifies its resources and requirements. Furthermore, a dynamic partitioning system named CloneCloud [6] formulates the dynamic partitioning problem and develops an execution runtime platform with flexible application partitioner to enable 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.

3. DECOMPOSITION GRANULARITY

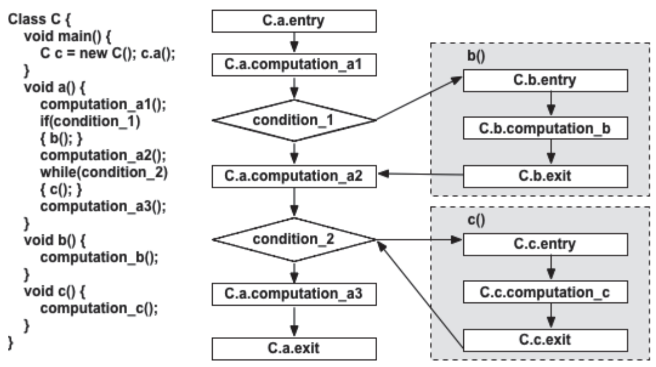
There are various of definition for "Decomposition" in computer sciences. In this paper, the term of decomposition is defined as "breaking a large system down into progressively smaller classes or objects that are responsible for some part of the problem domain". Decomposition is an prerequisite requirement of the cognitive decomposed cloud gaming system, since the cognitive 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. So how shall a game program be decomposed? In this section, we discussed the decomposition granularity and their challenges.

3.1. 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. 2.

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.



```
Class C {
  void main() {
    C c = new C(); c.a();
  }
  void a() {
    computation_a1();
    if(condition_1)
    { b(); }
    computation_a2();
    while(condition_2)
    { c(); }
    computation_a3();
  }
  void b() {
    computation_b();
  }
  void c() {
    computation_c();
  }
}
```

Fig. 2. Fine-Grained Decomposition Example

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 these 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 [6], 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.

3.2. Coarse-Grained Decomposition

In contrary 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 by a set of objects and methods, which work collaboratively within the scope of the component. See Fig. 3 for an illustration.

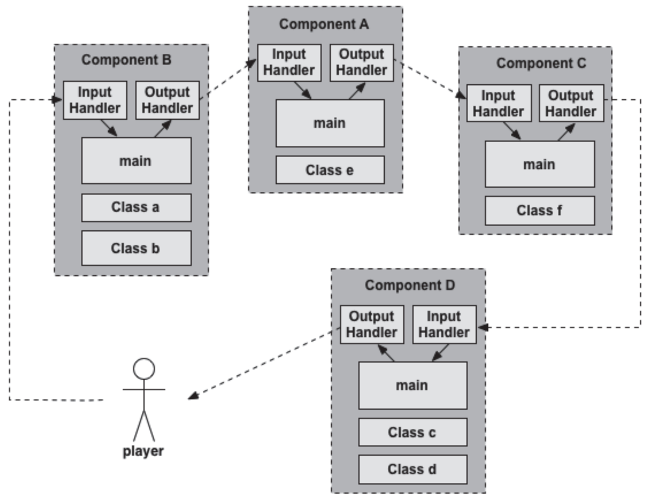


Fig. 3. Coarse-Grained Decomposition Example

In this program, the instance of *Component B* is activated by an control instruction from 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 priori and to pre-define all unified interfaces between the components.

4. PERFORMANCE EVALUATION

As the references for cognitive decisions on component allocation, the system shall perform real-time measurement on each component's execution performance in both cloud and terminals with various capacities. Moreover, since the invocation between components may occur remotely, the evaluation on the efficiency of message exchanges is also mandatory. In this session, we discuss both of the issues.

4.1. Execution Performance

For fine-grained decomposition, previous work [6] proposes an offline prediction on components' execution performances with *static analyzer* and *dynamic profiler*. In order to optimize the partitioning, the authors parses the invocation structures of the target program and applied static profiler data, generated by multiple executions of the application with a randomly chosen set of input data, to form a profile tree. However, to retrieve the profile output data (execution time and energy consumed in [6]), the system need to temporarily instrument application-method entry and exit points during each profile run on each platform, which is unfriendly requirement if the platform is open for developers. In addition, random inputs generated in the profile may not explore all relevant execution paths of the application. Therefore, the offline prediction will be even more inaccurate when the application is a video game, since it is a high-dynamic interaction system.

To overcome these restricts, we propose a statistical approach to predict the execution performance online. We design *Execution Monitor* to monitor resource usage of each components, whether in cloud or terminal, and save these execution information into a statistics database (example as shown in Table 1, including memory consumption, CPU usage percentage, execution time, output data size, execution environment, etc.

Table 1. 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
...	...	...	...	...	...	...

However, the information extracted from the execution information database is insufficient for the cognitive optimization, since it requires all components' execution performances in both cloud and terminal. Nevertheless, it is infeasible in real system to migrate all components to the terminals and evaluate their execution performance for optimization. Therefore, we implement a *Prober* component with a series of computational iterations and measure its execution information in both cloud and terminal. With this approach, the system is able to compare the computational efficiency of cloud and terminal, thus, estimate the execution performances for all components. 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 component 3 can be estimated as $20ms \times R_E$.

4.2. Communication Performance

The set of components are working in a collaborative manner in a decomposed program. Their cooperation involves information communications with messages. In conventional program design, inter-component communications, i.e. an invocation between methods, are always occurred in a local environment. However, for the decomposed cloud games, a simple communication might introduce a network transmission, given the two involving components are hosted in cloud and terminal, respectively. Therefore, to evaluate the communication performances between components are of great importance.

The execution information database introduced in Section 4.1 provides a statistic for all communications between components. With these information, the system constructs a communication tree of the game program, as depicted in Fig. 4. The edges' weight, which refers to their communication costs, are mainly subject to two aspects: communication data size and network quality of service (QoS) parameters. Com-

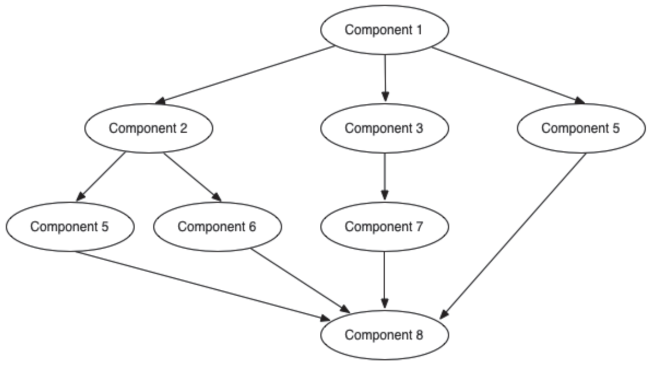


Fig. 4. A Component Communication Tree

munication data size can be simply measured in the entry and exit of the component, while the network QoS parameters are variables that need to be measured in real-time. Therefore, the *Prober* component proposed in Section 4.1 is reused as a network QoS tester. With known data size, its transmission cost is a perfect reference for all remote invocations with network communications.

5. RESPONSE DELAY

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 [7] segments response delay of a conventional RR-GaaS cloud gaming system into three components: network delay, processing delay and play out delay. However, to calculate the response delay in a decomposed cloud game is complete different. Take the communication tree in Fig. 4 as an example. 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. Therefore, we formulate the response delay d_c for response cycle c as:

$$d_c = \sum_{n \in c} e_n + \sum_{i \in c, j \in c} c_{i,j} \qquad (1)$$

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: $d_{max} = \max_{c \in C} d_c$. Hereby we denote the constraints of response delay on a decomposed cloud game as:

$$d_{max} \leq d_{tolerate} \qquad (2)$$

6. CONCLUSION

Decomposition is a fundamental requirement of cognitive resource allocation in cloud gaming systems. In fact, the decomposition architecture follows the distribution nature of cloud resource infrastructure, which also provides potential improvement on intra-cloud efficiency. In this paper, we present the design principles from the aspect of cloud game system and discuss their key challenges and research issue.

7. REFERENCES

[1] W. Cai, M. Chen, and V. Leung, "Towards gaming as a service," *IEEE Internet Computing*, May 2014.

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

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

[4] W. Cai and V. Leung, "Multiplayer cloud gaming system with cooperative video sharing," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp. 640–645.

[5] 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*, Berlin, Heidelberg, 2009, Middleware'09, pp. 83–102, Springer-Verlag.

[6] 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*, New York, NY, USA, 2011, EuroSys '11, pp. 301–314, ACM.

[7] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM International Conference on Multimedia*, New York, NY, USA, 2011, MM '11, pp. 1269–1272, ACM.