ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

CLOUD GAMING PROPOSES GPU/CPU HYBRID CLUSTER ON THE USER LEVEL VIRTUALIZATION

Sr.C.Jansi Sophia Mary[1], A.Kalaiselvi[2]
[1] Assistant Professor, Department of CSE,
Idhaya Engineering College for Women, Chinnasalem,
Sr.j.sofi@gmail.com

[2] PG Scholar, Department of CSE,
Idhaya Engineering College for Women, Chinnasalem,
kalaibecse22@gmail.com

ABSTRACT

The GCloud, a GPU/CPU hybrid cluster for cloud gaming based on the userlevel virtualization technology. Specially, we present a performance model to analyze the server-capacity and games' resource-consumptions, which categorizes games into two types: CPU-critical and memory io critical. Consequently, several scheduling strategies have been proposed to improve the resource utilization and compared with others. Simulation tests show that both of the First-Fit-like and the Best-Fit-like strategies outperform the others; especially they are near optimal in the batch processing mode. Other test results indicate that GCloud is efficient: An off-the-shelf PC can support five high-end video-games run at the same time. In addition, the average per frame processing delay is ms under different image resolutions, which outperforms other similar solutions.

Keywords - Cloud computing, cloud gaming, resource scheduling, user-level virtualization

I. INTRODUCTION

CLOUD gaming provides game-on-demand services over the Internet. This model has several advantages. it allows easy access to games without owning a game console or high end graphics processing units (GPUs); the game distribution and maintenance become much easier. For cloud gaming, the response latency is the most essential factor of the quality of gamers' experience "on the cloud". The number of games that can run on one machine simultaneously is another important issue, which makes this mode economical and then really practical. Thus, to optimize cloud gaming experiences, CPU / GPU hybrid systems are usually employed because CPU-only solutions are not efficient for graphics rendering. One of the industrial pioneers of cloud gaming, Onlive emphasized the former: it allocated one GPU per instance for high end video games. To improve utilization, some other service providers use the virtual machine (VM) technology to share the GPU among games running on top of VMs. stream games from cloud servers located around the world to internet-connected devices. Since the end of 2013, Amazon EC2 has also

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

provided the service for streaming games based on VMs. More technical details can be acquired from noncommercial projects. GamePipe is a VM-based cloud cluster of CPU/GPU servers. Its characteristic lies in that, not only cloud resources but also the local resources clients can be employed to improve the gaming quality. Another system, GamingAnywhere [3], has used the userlevel virtualization technology. Compared with some solutions, its processing delay is lower.

Besides, task scheduling is regarded as another key issue to improve the utilization of resources, which has been verified in the high performance GPU computing fields,. However, to the best of our knowledge, the scheduling research for cloud gaming has not received much attention yet. One example based on VMs is VGRIS including its successor VGASA. It is a GPU resource management framework in the host OS and schedules virtualized resource of guest OSes. This paper proposes the design of a GPU/CPU hybrid sytem for cloud gaming and its prototype, GCloud. GCloud haused the user level virtualization technology to implement sandbox for different types of games, which can isolate more than one game instance from each other on a game server transparently capture the game's video/audio outputs fostreaming, and handle the remote client-device's inputs. Moreover, a performance model has been presented thus we have analyzed resource consumptions of games and performance bottlenecks of a server, through excessive experiments using a variety of hardware performance counters. Accordingly, several task scheduling strategies have been designed to improve the server utilization and been evaluated respectively. [4] discussed about creating Obstacles to Screened networks. In today's technological world, millions of individuals are subject to privacy threats. Companies are hired not only to watch what you visit online, but to infiltrate the information and send advertising based on your browsing history.

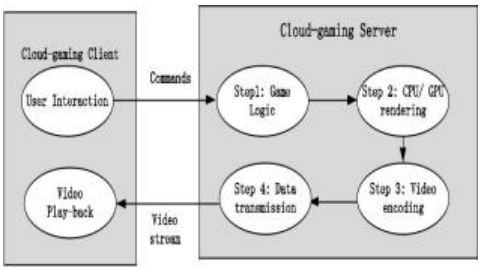


Fig.1. The whole workflow of cloud gaming.

First, it implements a virtual input-layer for each of con-currently-running instances, rather than a system-wide one, which can support more than one Direct-3D games at the same time. Second, it designs a virtual storage layer to trans-parently store each client's configurations across all servers, which has not been mentioned by related projects.

In summary, the following contributions have been accomplished:

[1] Enabling-technologies based on the light-weight virtu-alization are introduced, especially those of GCloud 's characteristics. (Section 3)

[2] To balance the gaming-responsiveness and costs, we adopt a "just good enough" principle to fix the FPS (frame per second) of games to an acceptable level. Under this principle, a performance model is con-structed to analyze resource consumptions of games, which categorizes games into two types: CPU-critical and memory-io-critical; thus several scheduling mech-anisms have been presented to improve the utiliza-tion and compared. In addition, different from previous jobs focused on the GPU-resource, our work has found the host CPU or the memory bus is the system bottleneck when several games are run-ning simultaneously. (Section 4)

[3] Such a cloud-gaming cluster has been constructed, which supports the mainstream game-types. Results of tests show that GCloud is highly efficient: An off-the-shelf PC can support up to five concurrently-run-ning video-games (each game's image-resolution is

1024 768 and the frame per second is 30). The aver-age per-frame processing delay is 8 19 ms under different image-resolutions, which can satisfy the stringent delay requirement of highly-interactive

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

games. Tests have also verified the effects of our performance model.

II. RELATED WORK

1 Cloud Gaming

Cloud gaming is a type of online gaming that allows direct and on-demand streaming of game-scenes to

networked-devices, in which the actual game is running on the server-end (main steps have been described in Fig. 1). Moreover, to ensure the interactivity, all of these serial operations must happen in the order of milliseconds, which challenges the system design critically.

The amount of latencies is defined as interaction delay. Existing researches [10] have shown that different types of games put forward different requirements.

One solution type of cloud-gaming is VM-based. For the solutions based on VMs, Step 1 is completed in the guest OS while others on the server-end are accomplish by the host. Barboza et al. [11] presents such a solution, which provides cloud gaming services and uses three levels of managers for the cloud, hosts and clients. Some existing work, like GaiKai, G-cluster, Amazon EC2 for streaming games and GamePipe [2], also belong to this category.

In contrast to VM-based solutions, the user-level solution inserts the virtualization layer between applications and the run-time environment. This mode simplifies the processing stack; thus it can reduce the extra overhead. GamingAnywhere [3] is such a user-level implementation, which sup-ports Direct3D/SDL games on Windows and SDL games on Linux.

Some solutions have enhanced the thin-client protocol to support interactive gaming applications. Dependent on the concrete implementation, they can be classified into the two types. For example, Winter et al. [12] have enhanced the thin-client server driver to integrate a real-time desktop streamer to stream the graphical output of applications after GPU processing, which can be regarded as a light-weight virtualization-based solution. In contrast, Muse [13] uses VMs to isolate and share GPU resources on the cloud-end, which has enhanced the remote frame buffer (RFB) protocol to compress the frame-buffer contents of server-side VMs.

However, these researches have focused on the optimization of interaction delay, namely, taken care of the performance of a single game on the cloud, rather than the interference between concurrently-running instances. Moreover, none of these systems has presented any specific scheduling strategy.

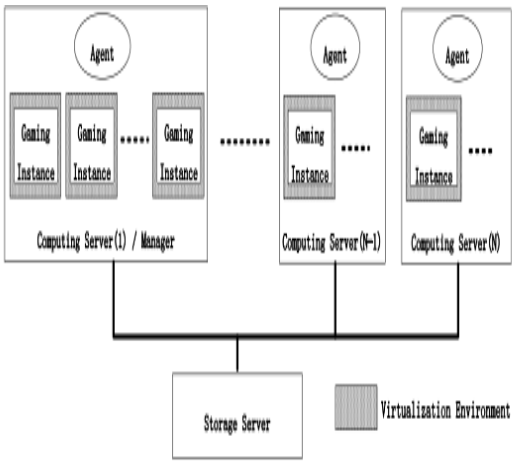


Fig 2.System architecture

SYSTEM ARCHITECTURE AND ENABLING TECHNOLOGIES

3.1 The Framework

The system (in Fig. 2) is built with a cluster of CPU / GPU-hybrid computing servers; a dedicated storage server is used as the shared storage. Each computing server can host the execution of several games simultaneously. One of these servers is employed as the manager-node, which collects real-time running information of all servers and completes management tasks, including the task-assignment, user authentication, etc.

It is necessary to note that the framework in Fig. 2 is for small / medium system-scales. For a large scale system with many users, a hierarchical architecture is needed to avoid the

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

bottleneck of information-exchange. In fact, because the quality of gamers' experience highly depends on the response latency and the latter is sensitive to the physical distance between clients and servers, the architec-ture may be geographically-distributed, which is out of scape of this paper. It also means that in one site the scale will not be very large.[5]

Initially, gaming-agents on available computing servers register to the manager, indicating that they are ready and.which games they can execute. When a client wants to play some game, the manager will search for candidates among the registered information. After such a server has been choosen, a start-up command will be sent to the corresponding agent to boot up the game within a light weight virtualization environment. Then, its address will be sent to the client. Future communication will be done directly between the two ends.

During the run time, each agent collects local runtime information and sends it to the manager periodically; the latter can get the latest status of resource-consumptions.

The storage server is an important role to provide the personalized game-configuration for each user. For instance, User A had played Game B on Server C. Now A wants to play the game again while the manager finds that Server C's resources have been depleted. Then the task has to be assigned to another server, D. Consequently, it is necessary to restore A's configurations of B on D, including the game's progress and other customized information. The storage server is just used as the shared storage for all computing nodes.

III. SYSTEM MODEL

3.2.1 Image Capture

Usually, gaming applications employ the mainstream 3D computer-graphics-rendering libraries, like Direct3D or OpenGL, to complete the hardware (GPU) acceleration; GCloud supports both of them.

In the case of Direct3D, the typical workflow of a game is usually an endless loop: First, some CPU computation pre-pares the data for the GPU, e.g., calculating objects in the upcoming frame. Then, the data is uploaded to the GPU buffer and the GPU performs the computation, e.g., render-ing, using its buffer contents and fills the front buffer. To fetch contents of the image into the system memory for the consequent processing, we intercept the Direct3D's Present API.

For OpenGL, we have intercepted the Present-like API in OpenGL, glutSwapBuffers, to capture images.

For other games based on the common GUI window, we just set a timer for the application's main window, then we intercept the right message handler to capture the image of the target window periodically.

3.2.2 Audio Capture

Capturing of audio data is a platform-dependent task. Because our main target platform is MS Windows, we inter-cept Windows Audio Session APIs to capture the sound. Core Audio serves as the foundation of quite a few higher-level APIs; thus this method can bring about the best adaptability.

3.2.3 Virtual Input Layer

Flash-based or OpenGL-based applications are usually using the window's default message-loop to handle inputs. Thus, the solution is straightforward: We inject a dedicated input-thread into the intercepted game-process. On recep-tion of any control command from the client, this thread will convert it into a local input message and send it to the target window.

For Direct3D-based games, the situation is more compli-cated. The existing work [3] replays input events using the SendInput API on Windows. However, SendInput inserts events into a system-wide queue, rather than the queue of a specific process. So, it is difficult to support more than one instance for the non-VM solution. To conquer this problem, we intercepted quite a few DirectInput APIs to simulate input-queues for any virtualized application; thus the user's input can be pushed into these queues and made accessible to applications.

3.2.4 Virtual Storage Layer

From the storage aspect, a program can be divided into three parts [31]: Part 1&2 include all resources provided by the OS and those created/modified by the installation pro-cess; Part 3 is the data created/modified/deleted during the run time, which contains game-configurations of each user. For the immutable parts, it is relatively easy to distribute them to servers through some system clone method. The focus is how to migrate

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

resources of Part 3 across servers to provide personalized game-configurations for users.

We construct a virtual storage layer by the interception of file-system and registry accessing APIs of all games. During the run time, the resource modified by the game instance will be moved into Part 3. When the previously-described case in Section 3.1 occurs, the virtual storage layer of Game B on the current server can redirect resource-accesses to the shared storage to visit the latest configurations of User A, which were stored by the last run on Server C.

IV. SCOPE OF RESEARCH

The response latency and the number of games that one machine can execute simulta-neously are both essential to a cloud gaming system. To a large extent, they are in contradiction and existing systems (like [3], [11], [12]) usually focus on the first issue.

However, it is not always economical. For example, if the FPS of a given game is too high, it will consume more resources. Moreover, the loss compression will counteract the high video-quality to a certain extent.

Some scheduling work, like VGRIS / VGASA [8], [9], has presented multi-task scheduling strategies. There are several essential differences between our work and VGRIS / VGASA: First, they are focused on how to schedule existing games on a server, including the allocation of enough GPU resources for a game, etc. In contrast, GCloud is focused on the assignment of a new task. Second, they are focused on the GPU resource and no any other operation (like image-capture, encoding, etc.) has been considered, while our tests (presented in Section 4.4) show the host CPU or the memory bus is the bottleneck. Third, VGRIS and VGASA are VM-specific.

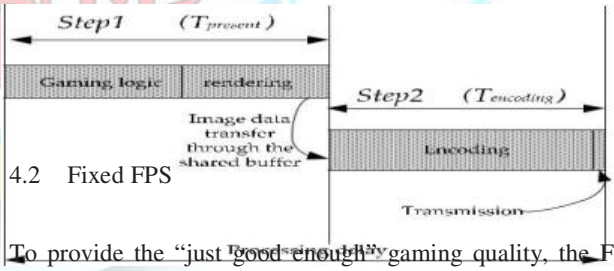
4.1 Game Quality

A cloud gaming system's interaction delay contains three parts [27]: (1) Network delay, the time required for a round of data exchange between the server and client; (2) Play-out delay, the time required for the client to handle the received for playback; (3) Processing delay, required for the server to process a player's command, and to encode and send the corresponding frame back.

This paper is mainly about the server-side and the net-work is assumed to be able to provide the sufficient band-width, thus we focus on the processing delay that should be confined into a limited range. The work [25] on measuring the latency of cloud gaming has disclaimed that, for some existing service-providers (like Onlive), the processing delay is about 100-200ms. Thus, we use 100 ms as our scheduling target, denoted MAX_PD. Another measurement of key metrics is FPS; the required FPS is illustrated as FIXED_FPS. In this work, FIXED_FPS is set to 30 by default.

The gaming workflow can be regarded as a pipeline including four steps: operations of gaming logic, graphic rendering (including the image cap-ture), encoding (including the color-space conversion) and transmission. In addition, our tests show that given the suf-ficient bandwidth, the delay of transmission is much less than other steps. Thus, the fourth step can be skipped and we focus on the remaining three.Furthermore, the first two steps are completed by the intercepted process, which is transparent to us; thus we should combine them together and the sum of these laten-cies is denoted by $T_{present}$. The average processing time of the encoding step is denoted by $T_{encoding}$ (The pipeline is presented in Fig. 3). Hence, if the following conditions (referred as Responsiveness Conditions) have been satisfied, the requirement on the FPS and processing delay will be met undoubtedly. To be more precise, satisfaction of the first two conditions means the establishment of the last one, under the default case.

$T_{present}$ < ¼ 1=FIXED FPS



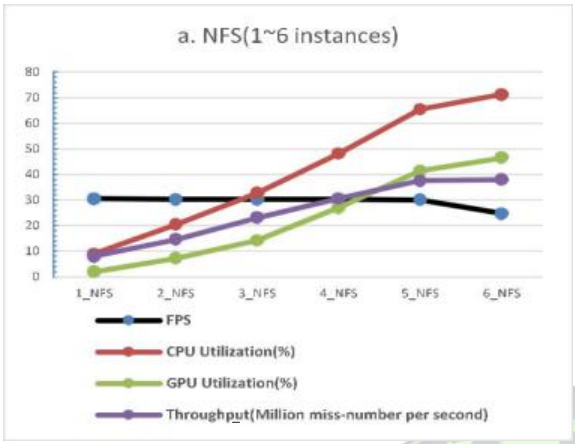
4.2 Fixed FPS

To provide the "just good enough" gaming quality, the FPS value should be fixed to some acceptable level (Issue 1). Because the interface of GPU drivers is not open, our solu-tion is in the user-space, too.

Take the Direct3D game as an example, we intercept the Present API to insert a Sleep call for adjusting the loop latency: The rendering complexity is mostly affected by the complexity of gaming scenes and the latter changes gradu-ally. Thus, it is reasonable to predict $T_{present}$ based on its own historical information. In the implementation, the aver-age time (denoted $T_{avg\ present}$) of the past 100 loops is used as the prediction for the upcoming one (the similar method

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017



has been adopted by [8], [9]) and the sleep time (T_{sleep}) is calculated as:

$$T_{sleep} < ¼ 1=FIXED\ FPS \quad T_{avg\ present}$$

The true problem lies in how to judge whether a busy server is suitable to undertake a new game instance or not. Thus, we should solve Issue 2 anyway.

4.3 Hardware-Assistant Video Encoding

The fixed-FPS can mitigate the inference between games because it allocates just enough resource for rendering. Further, we use the hardware-assistant video-encoding capabil-ity of commodity CPUs for less inference.

The hardware technology of Intel CPUs, Quick Sync, has been employed. It owns a full-hardware function pipeline to compress raw images in the RGB or YUV format into the H264 video. Now Quick Sync has become one of the mainstream hardware encoding technologies.[6] On the test server, a Quick-Sync-enabled CPU can simultaneously support up to twenty 30-FPS encoding tasks (the image resolution is 1024 768); the latency for one frame is as low as 4.9 ms.

4.4 Resource-Metrics

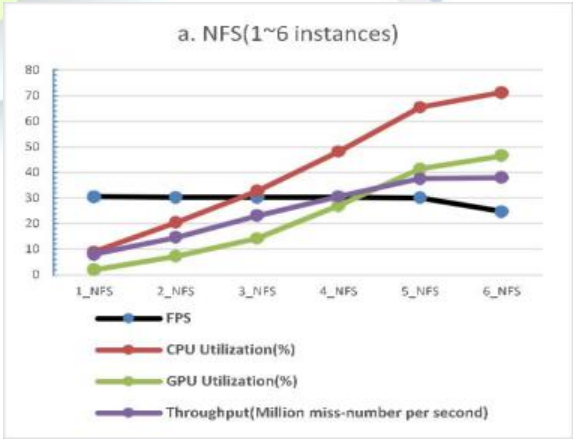
Five types of system-resources have been focused on, including the CPU, GPU, system-RAM, video-RAM and the system bandwidth: The first two can be denoted by utiliza-tion ratios; the next two are represented by memory con-sumptions and the last refers to the miss number of the LLC (Last Level Cache). Correspondingly, the server capacity and the average resource requirements of a game (under the condition satisfying the Responsiveness Conditions) can be denoted by a tuple of five-items, <U_CPU, U_GPU, M_HOST, M_GPU, B>.

4.4.1 Test Methods

Commercial GPUs usually implement driver / hardware counters to provide the runtime performance information. For example, the NVIDIA's PerfKit APIs[7] can collect resource-consumption information of each GPU in real time. Hence, we can get results accumulated from the previ-ous time the GPU was sampled, including the percentage of time the GPU is idle/busy, the consumption of graphic memories, etc.

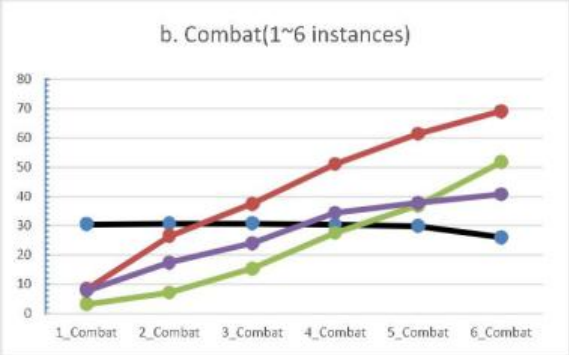
For commodity CPUs, the similar method has been used, too. For instance, Intel has already provided the capability to monitor performance events inside processors. Through its performance counter monitor (PCM), a lot of perfor-mance-related events per CPU-core, including the number of LLC-misses, instructions per CPU cycle, etc., can be obtained periodically.

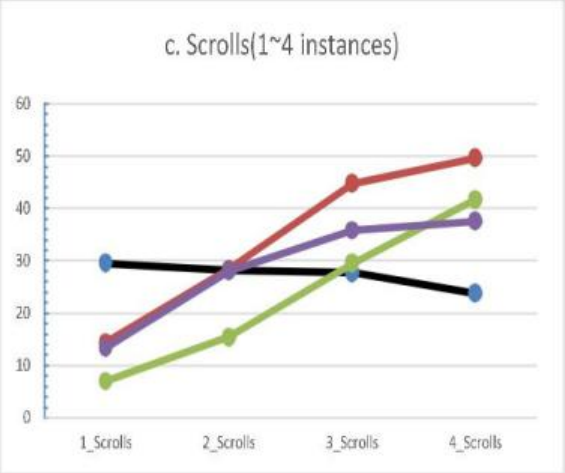
The sample periods for CPU and GPU are both set to 3s. In addition, we embed monitoring codes into the inter-cepted gaming APIs to record processing delays of each frame, which will be used to judge whether the Responsive-ness Conditions have been met or not.

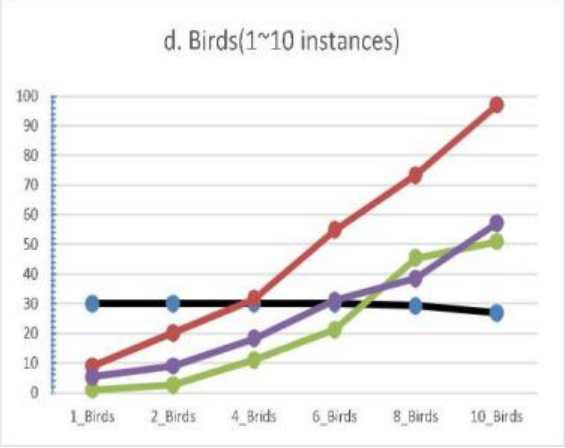


ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017







5. EXPERIMENTAL RESULTS

The test environment and configurations are the same as those in Section 4.4, as well as the testing method.

5.2.1 Overheads of the User-Level Virtualization Technology Itself

We execute a game on a physical machine directly and record the game speed (in term of the average FPS) and average memory consumption. Then, this game is running in the user-level virtualization environment (all related APIs have been intercepted but no any real work, like image capture, encoding, etc., has been enabled) and in a virtual machine respectively; the same runtime information will be recorded repeatedly.

5.2.2 Processing Performance of the Server

The processing procedure of a cloud-gaming instance can be divided into four parts: (1) image capture, which copies a rendered into the system memory, (2) video encoding,(3) tferring, which sends each compressed-frame into the network, and (4) the process of the

5.2.3 Multiple Games

The “just good enough” strategy is used; a Sleep call has been used to fix the FPS. First, an OpenGL game and three Direct3D games have been played one by one and the processing delay (including the sleep time) is sampled periodically; the sample period is one frame. Second, quite a few game combinations, each including more than one game, have been executed and sampled. Without loss of general-ity, FPS values of some game combinations that are played simultaneously are presented in Table 4, as well as the average absolute deviations (AADs). These combinations are:

Case 1: Two NFS instances;

Case 2: One NFS, one Combat and one Scrolls;

Case 3: Two NFS, one Combat and one Scrolls;

Case 4: One NFS, one Combat, one Scrolls and two Birds.
On the whole, the average FPS ranges from 30.5 to 31.5 as

5.2.4 Verification of the Performance Model

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

According to the result of the performance model and scheduling strategy, we test several typical server loads for verification. Without loss of generality, the following cases have been presented.

1) One Scrolls, one Combat and two NFS. As presented in Table 5 (1st row), the FPS value of each game is more than 27 and the lowest is Scrolls's, about 27.1. All are not less than 90 percent of the FIXED_FPS (30), thus they are accepetable. Because the system-RAM band-width has been nearly exhausted (about 93 percent of the MAX_SYSTEM_BANDWIDTH), when another game join (regardless NFS or Birds), the FPS of Scrolls will drop below the acceptable level.

5.2.4 Discrepancy between Video and Audio

The delay fluctuations of games. The corresponding FPS-values will be less than 30, which will increase the timing discrepancy, because the accumulation process of audio-data will be slowed.

The network's delay fluctuations. They will increase the timing discrepancy, too. Our tests are carried out in the campus. We believe, for the Internet, this fac-tor will cause more delays.

The measurement error. The recording software records the screen periodically, 30 FPS, while the audio recording is consecutive. Thus, beginnings of some sequences of full-black images may be lost, which will decrease the gap.

VI. CONCLUSION

GCloud, a GPU/CPU hybrid cluster for cloud gaming based on the user-level virtualization technology. We focus on the guideline of task scheduling: To balance the gaming-responsiveness and costs, we fix the game's FPS to allocate just enough resources, which can also mitigate the inference between games. Accordingly, a per-formance model has been analyzed to explore the server-capacity and the game-demands on resource, which can locate the performance bottleneck and guide the task-sched-uling based on games' critical resource-demands. Compari-sons show that both the First-Fit-like and Best-Fit-like scheduling strategies can outperform others. Moreover, they are near optimal in the batch processing mode.

REFERENCES

[1] R. Shea, L. Jiangchuan, E.C.-H. Ngai, and C. Yong, "Cloud gam-ing: Architecture and performance," IEEE Netw., vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

[2] Z. Zhao, K. Hwang, and J. Villeta, "GamePipe: A virtualized cloud platform design and performance evaluation," in Proc. ACM 3rd Workshop Sci. Cloud Comput., 2012, pp. 1–8.

[3] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in Proc. ACM Multimedia Syst., Feb. 2013, pp. 36–47.

[4] Christo Ananth, P.Muppidathi, S.Muthuselvi, P.Mathumitha, M.Mohaideen Fathima, M.Muthulakshmi, "Creating Obstacles to Screened networks", International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST), Volume 1,Issue 4,July 2015, pp:10-14

[5] V. T. Ravi, M. Becchi, G. Agrawal, and S. T. Chakradhar, "Supporting GPU sharing in cloud environments with a transpar-ent runtime consolidation framework," in Proc. 20th ACM Int. Symp. High Perform. Distrib. Comput., 2011, pp. 217–228.

[6] G. A. Elliott and J. H. Aon, "Globally scheduled real-time multiprocessor systems with GPUs," Real-Time Syst., vol. 48, no. 1. pp. 34–74, 2012.

[7] L. Chen, O. Villa, S. Krishnamoorthy, and G. R. Gao, "Dynamic load balancing on single- and multi-gpu systems," in Proc. IEEE Int. Symp. Parallel Distrib. Process., 2010, pp. 1–12.

[8] M. Yu, C. Zhang, Z. Qi, J. Yao, Y. Wang, and H. Guan, "GRIS: Virtualized GPU resource isolation and scheduling in cloud gaming," in Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput., 2012, pp. 203–214.

[9] C. Zhang, J. Yao, Z. Qi, M. Yu, and H. Guan, "vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 11, pp. 3036–3045, 2014.

[10] M. Claypool and K. Claypool, "Latency and player actions in online games," Commun. ACM, vol. 49, no. 11, pp. 40–45,2006