Cloud-Based Gaming Service Platform Supporting Multiple Devices

Kyoung Ill Kim, Su Young Bae, Dong Chun Lee, Chang Sik Cho, Hun Joo Lee, and Kyu Chul Lee

To implement a cloud game service platform supporting multiple users and devices based on real-time streaming, there are many technical needs, including game screen and sound capturing, audio/video encoding in real time created by a high-performance server-generated game screen, and real-time streaming to client devices, such as low-cost PCs, smart devices, and set-top boxes. We therefore present a game service platform for the running and management of the game screen, as well as running the sound on the server, in which the captured and encoded game screen and sound separately provide client devices through real-time streaming. The proposed platform offers Web-based services that allow game play on smaller end devices without requiring the games to be installed locally.

Keywords: Game service platform, game on demand, audio/video encoding, real-time streaming, cloud-based game.

Manuscript received Apr. 9, 2013; revised Sept. 24, 2013; accepted Oct. 12, 2013.

This work was supported by the IT R&D program and the Game Service Technology Based on Realtime Streaming (10035184) funded by the Ministry of Knowledge Economy (MKE), Rep. of Korea.

Kyoung Ill Kim (phone: +82 42 860 6467, kki@etri.re.kr), Su Young Bae (manim75@etri.re.kr), Dong Chun Lee (bluepine@etri.re.kr), Chang Sik Cho (cscho@etri.re.kr), and Hun Joo Lee (hjoo@etri.re.kr) are with the Creative Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Kyu Chul Lee (kclee@cnu.ac.kr) is with the Department of Computer Engineering, Chungnam National University, Daejeon, Rep. of Korea.

I. Introduction

A cloud game service platform supporting multiple users and devices based on real-time streaming performs the capturing of the game screen and sound separately, encoding in real time created by a high-performance server-captured game screen with sound and real-time streaming to the clients. On the client side, the user interactive information from client input devices, such as a keyboard, mouse, and joystick, is streamed to the server. Cloud computing has recently emerged as a business model enabling information technology infrastructure and software to be delivered directly over the Internet as a service. In other words, cloud computing is a general term for complete services delivered over networks using self-service end-user portals and is a flexible business model that incorporates hardware, software, and services into a single revenue stream.

One of the latest advancements in real-time game streaming technology that enables such ubiquitous gaming is a cloud-based gaming service system known as game on demand (GoD) [1] service. GoD is a networked media platform that offers Web-based service, allowing game play on multiple devices, such as low-cost PCs, smart devices, and set-top boxes, without locally installed games. For these reasons, cloud-based game services have many advantages in that users can play high-quality games, such as console games and massively multiplayer online role-playing games (MMORPGs), at any time and anywhere, regardless of their client device capabilities. In addition to the game maker's point of view, GoD can easily secure a number of users and arrange a variety of revenue-based services. It can also prevent financial losses from an infringement of copyright in the gaming market.

Real-time interactive streaming technology for GoD service and encoding/streaming technology for supporting multiple

http://dx.doi.org/10.4218/etrij.13.2013.0076

users and devices require a large amount of streaming services, such as video on demand and software as a service. In particular, large numbers of high-quality streaming service technologies may resolve the data security issue and a loss of distributed computing environment occurring in the acquisition and management of PCs.

In this paper, we propose a new cloud-based game service platform supporting multiple users and devices based on real-time streaming technology. To provide convincing real-time streaming based on the game service platform, we can describe the key requirements of cloud-based game service systems as follows:

- Under network congestion, such network problems as increased latency, jitter, and packet losses are distributed evenly in all competing traffic. However, the quality can be enhanced using quality of service (QoS) technologies to give higher priority to game traffic during network bottlenecks [2].
- Latency time is defined as the time between a player's key action and the time of the actual resulting game output on the player's screen. Since computer games are highly interactive, an extremely low latency time has to be achieved. Typically, the interaction latency time in an online gaming service does not exceed 200 ms.
- A number of games are operated on game servers at the same time, and it is therefore hard to manage information between the game server and clients. Therefore, input/output device virtualization is a methodology used to simplify management, lower costs, and improve the performance of game servers in an enterprise environment.

The rest of the paper is organized as follows. In section II, we briefly survey previous works on GoD and audio/video encoding for cloud-based gaming services. We describe the proposed system architecture and the core systems used in our gaming service platform in section III. In section IV, we explain the implementation details of our gaming service platform supporting multiple users and devices based on real-time streaming and describe the encoding/streaming performance results. Finally, we discuss future works and offer some concluding remarks in section V.

II. Related Works

In this section, we provide a brief overview of other work related to cloud-based gaming service technologies. We note their many technical characteristics and the audio/video encoding methods used, especially those designed for the real-time streaming video game market.

GoD service has a number of commercial systems that have emerged in the market [2]. OnLive is an on-demand gaming entertainment platform that was announced at the Game Developers Conference in 2009 [3]. Gaikai launched a GoD service based on cloud gaming technology that allows users to play major PC and console games [4]. The clients of their service can display audio/video game streams, streaming from the cloud, using previously installed plug-ins, such as Java or Adobe Flash, on the client devices. Even though both are cloud-based gaming services, OnLive and Gaikai have different goals in mind. OnLive sells full games, provides demos and Brag Clip videos, and allows users to watch other players play via the Arena view, whereas Gaikai advertises demo games through its Webpage [3]. The Games@Large framework enables commercial video games to stream from a local server to remote end devices in local area networks [5]. This system and its streaming protocols were developed and adapted for highly interactive video games [1], [6]. Ubitus cloud computing technology [7] adopts a distributed service-oriented architecture to accelerate the vast computing tasks efficiently within the cloud, whether they are multimedia conversions, game frame compressions, or other computational-intensive jobs. The cloud itself consists of heterogeneous platforms empowered by our unique algorithm that could distribute computing jobs evenly and intelligently across the network.

There are two major approaches for game streaming. One is a 3D graphics streaming approach exploited for streaming the game's output to directly transmit the graphics commands to the client device and render the image on a client device [8]. For the purpose of reducing server overhead and providing a stable service at a low cost, a client rendering method was introduced for desktop virtualization service in 3D graphics applications [9]. In this method, the host rendering commands are delivered to the client through the network and are executed by the client's graphics device. The other approach is video streaming in which the server renders the game graphics scene, the frame buffer is captured and eventually downsampled to match the target device resolution, and the current image is encoded using standard video codecs, such as MPEG-2, MPEG-4, and H.264 [10], [11]. Video streaming is intended for thin-client devices lacking hardware-accelerated rendering capabilities [12]. In our research, we exploit a real-time audio/video streaming method since our system should support multiple users and devices, such as low-cost PCs, smart devices, and set-top boxes.

In a real-time streaming game service, one of the key issues is the video compression method. OnLive introduced an interactive video compression method designed specifically for video games. To achieve high-performance encoding, they developed two types of dedicated compression hardware for video encoding: an optimized compressor based on human perception and a live compressor similar to a conventional

Table 1. Comparison of our cloud-based game service platform with other cloud-based game platforms.

		Platform			
		OnLive [3]	Ubitus [7]	Gaikai [4]	Proposed
Function	Platform	PC, Mac, micro console (TV), smartphone	PC, smartphone	PC, smartphone	PC, smartphone, MID, IPTV
	Portability	No	-	No	Yes (script download)
	Encoding resolution	720 p @ 60 fps	-	-	1080 p @ 60 fps
	Client support	2	2	-	8
	Virtualization	OS virtualization	-	-	App virtualization
	Encoding format	- (H/W)	-	H.264 (S/W)	H.264 (S/W)
	Support GPU	No	No	No	Yes (NVIDIA CUDA)

Note. -: detail specs are not announced.

compressor [3]. Recently, OTOY introduced a new video encoding method called ORBX. ORBX has been designed from the ground up to take advantage of OpenCL-based graphics processing unit (GPU) servers (FRC). ORBX encodes video entirely on the GPU, with at least 30 to 100 times the scaling of H.264 encoding solutions requiring either a CPU or specialized encoding ASIC. Unfortunately, technical information of the ORBX encoding method is not publicly available. NVIDIA's Kepler GPU includes a high-performance H.264 encoding engine capable of encoding simultaneous streams with superior quality. This provides a giant leap forward in cloud server efficiency by offloading the CPU from encoding functions and allowing the encode function to scale with the number of GPUs in a server [13].

The NVIDIA Kepler GPU architecture was announced at the Graphics Technology Conference in 2012 [14]. NVIDIA's new Kepler GPU-based VGX boards are specifically designed to enable rich graphics in visualized environments. This specifically addresses the three key barriers that have prevented earlier generations of GPUs from being practically implemented in cloud servers: GPU virtualization, low-latency remote display, and data center class power efficiency. Together, these technologies create a significant disruption in the performance, efficiency, and end-user experience delivered from cloud-based VDI services [13].

Table 1 shows a comparison of our cloud-based game service platform with other cloud game platforms. As shown in this table, the biggest difference between our cloud-based game service platform and existing cloud-based game platforms is in the portability. Previously developed online games cannot run in a cloud-based game environment. So, these games are modified or newly developed to run on the cloud-based game platforms. Because the main function of our cloud-based game platform is audio/video capturing, encoding and streaming to the clients running on the game server concurrently, the proposed platform offers Web-based services allowing game play on smaller end devices without requiring the games to be installed locally.

III. System Architecture

In this section, we propose the system architecture for a cloud game service platform supporting multiple users and devices based on real-time streaming. Our cloud gaming service platform consists of three major blocks: a Distributed Service Platform (DSP), a Distributed Rendering System (DRS), and an Encoding/Streaming QoS System (EQS).

1. System Overview

The DSP, a cloud-based gaming service platform based on the real-time streaming environments, consists of a DSP server and a number of DSP nodes and provides streaming game services to massively multiple users. The DSP server controls the multiple DSP nodes, efficiently distributes server resources to users who want to acquire the game services, and manages the shared resources between multiple users. In addition, DSP handles the DRS block, EQS block, and user input block for the game streaming and also comprehensively manages the whole system.

Figure 1 shows a task flow diagram and overview of the working scenarios as follows. The client accesses the DSP server (1.); the client sends its device information to the DSP server and waits for the allocation of the appropriate DSP server (2.). The DSP server that has received device information from the client looks for the DSP node that can provide the appropriate streaming services to the client based on the DSP node status. The DSP sever also transfers the selected DSP node's URL to the client (3.). When the client requests a login to the DSP node using the received URL, the game streaming service starts. After the client connects to the DSP node, the DSP node is responsible for processing the game input of multiple users through the UDP from client devices, such as a keyboard, mouse, and joystick. On the client side, multiuser game input is captured and transmitted through the UDP by the user input capture and transmission software on the client devices. In our system, several users connect to

ETRI Journal, Volume 35, Number 6, December 2013
http://dx.doi.org/10.4218/etrij.13.2013.0076

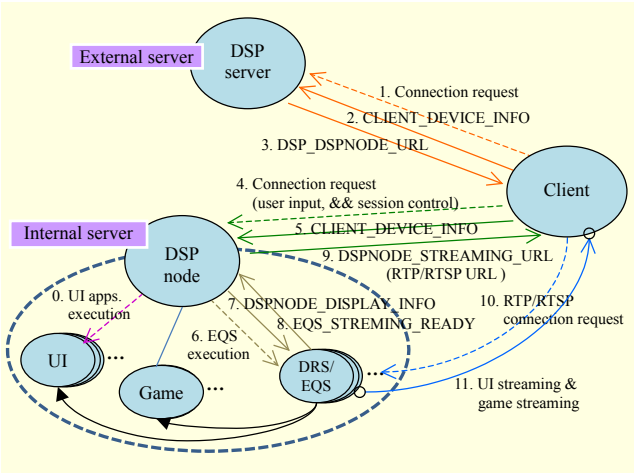


Fig. 1. Task flow diagram and working scenario.

Table 2. Resolutions and concurrent users.

Mode	Resolution	Connected user
VGA	640×480	8
SVGA	800×600	6
XGA	1024×768	4
HD	1280×720	4
FHD	1920×1280	1

the game server running several games simultaneously depending on the client device resolution.

Table 2 shows the number of maximum connected users running on the game server at the same time according to the client device resolutions. Our system supports five service modes for real-time streaming according to the resolutions of the client devices with a certain maximum number of users, as shown in Table 2. Here, we define the service mode, *SM*, which is determined by the resolution of the client devices, *r*, and the maximum connected users, *n*. Thus, *SM* can be defined as function **f**, which has *r* and *n* as the parameters for real-time game streaming services.

$$SM = \mathbf{f}(r, n).$$

The sound of a game has an important role in providing immersive gaming services along with visual elements. To support multiple users for each different game on a single server simultaneously, our system should support game streaming for each game. However, conventional audio processing based on dedicated hardware is difficult to directly exploit in multiple game streaming. The main problem of dedicated audio processing hardware in our configuration is the blending of multiple sounds for different games. Thus, we exploit a virtualization technique based on a virtual audio device for isolating multiple sounds.

First, we allocate a unique predefined index to a virtual audio device from 1 to 8 corresponding to the number of maximum connected users, *n*. Our system then configures the basic sound output device in a Windows system for multiple game sounds played through the virtual audio devices. Finally, the system captures the sounds for several games to encode the game streams into the AAC and H.264 formats.

The DRS is responsible for rendering a 3D scene and multi-view rendering for multiuser games [15]. To improve the 3D rendering performance in games, we utilize a multithreaded game engine [16] designed to scale to as many processors as are available within a platform.

The EQS is responsible for a multiuser game screen and sound capturing separately in real time, encoding the game audio/video running on the game server, and streaming the interactive game contents to the clients. To perform real-time streaming, the game audio/video streams to the clients, and there are two different methods used to capture video frames. In private games, which consist of game contents created for our system, the DSP requests the capturing of a rendered frame buffer for video encoding. In legacy games, which consist of game contents already used in online games, the DSP requests capturing the screen frame buffer running on the game server. In the rest of this paper, we describe the major functions in the EQS, including capturing the audio/video streams of multiple games, encoding, and streaming for the captured game audio/video streams.

2. EQS

The EQS consists of three major module components: capture module, encoding module, and streaming QoS module. Figure 2 shows an EQS functional flow between a game server and clients.

A. Audio/Video Capture

After the game server types and virtual sound driver are defined, the EQS is ready to accept the client request for capturing, encoding, and streaming to the clients. As shown in Fig. 2, new client requests join the game server, and the DSP assigns a suitable game server according to the resolution, memory state, number of connected users, and network state. We use shared memory on the inside with a queue to the receiving application that allows the use of PCM data buffering separately.

The video capture module is responsible for capturing the video running on the game server screen. To capture the game video frames for the private games, which consist of game contents created for our system, and for the legacy games,

which consist of game contents already used in online games, we have two different methods. For a private game, the DSP requests video that captures the rendered frame buffer from the DRS. For a legacy game, the DSP requests video that captures the frame buffer running on the game server. To implement visual capturing for the game frames, we utilize the DirectDraw primary surface to capture up to full HD (FHD) (1920×1080) in real time.

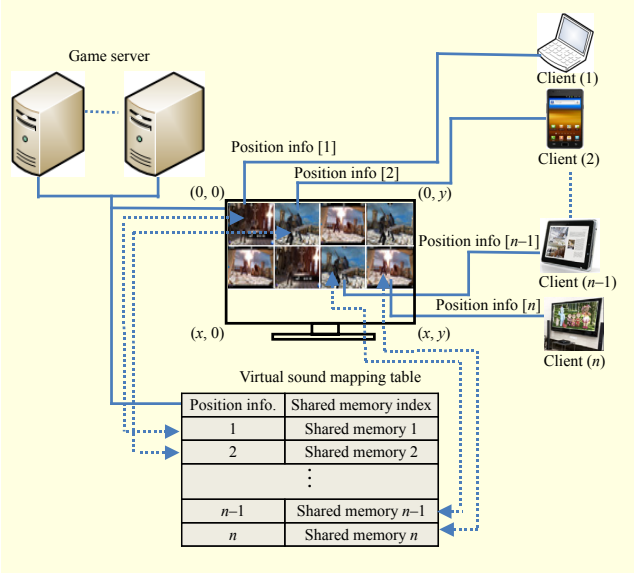


Fig. 2. EQS function flows.

Table 3. Features of audio/video encoding.

Supported feature	Audio	Video
Codec	HE-AACv2	x264, CUDA264
Resolution	-	320×240 to 1920×1080
Bitrate	16 kbps to 64 kbps	384 kbps to 5 Mbps
Frame rate	-	15 fps to 30 fps
Sampling rate	22.05 kHz to 48 kHz	-
Channel	Mono, stereo	-

B. Audio/Video Encoding

In our work, HE-AACv2 is utilized for audio streaming. HE-AACv2 was also standardized under the name Enhanced aacPlus by 3GPP for 3G UMTS multimedia services in September 2004. It is based on the AAC LC, SBR, and Parametric Stereo coding tools defined in the MPEG-4 Audio standard [17]. The EQS then transmits the encoded audio/video streams to the client using the Real-Time Transport Protocol (RTP) or the Real-Time Streaming Protocol [18], [19].

We develop the x264 and CUDA264 encoders to achieve low-delay video encoding of the captured game screen. Before the EQS performs the x264 encoding, a color-space conversion from RGB to YUV takes place on the captured frames. We utilize the 4:2:0 format for the YUV sampling to achieve the reduction of the storage data. Our video codecs cannot use the B-frame due to the real-time transfer and encoding delay. We also capture and encode the audio data for the games to transmit the interactive game contents to the clients. Table 3 shows the supported features of audio/video encoding in the proposed system.

C. RTP Streaming

We develop the RTP packetization using the rfc3984. The RTP payload format allows for a packetization of one or more Network Abstraction Layer Unit, produced by an x264 video encoder, in each RTP payload [18], [19]. The payload format has wide applicability, as it supports applications from simple low bitrate conversational usage, to high bitrate cloud-based gaming service.

On the other hand, the game client side devices for our

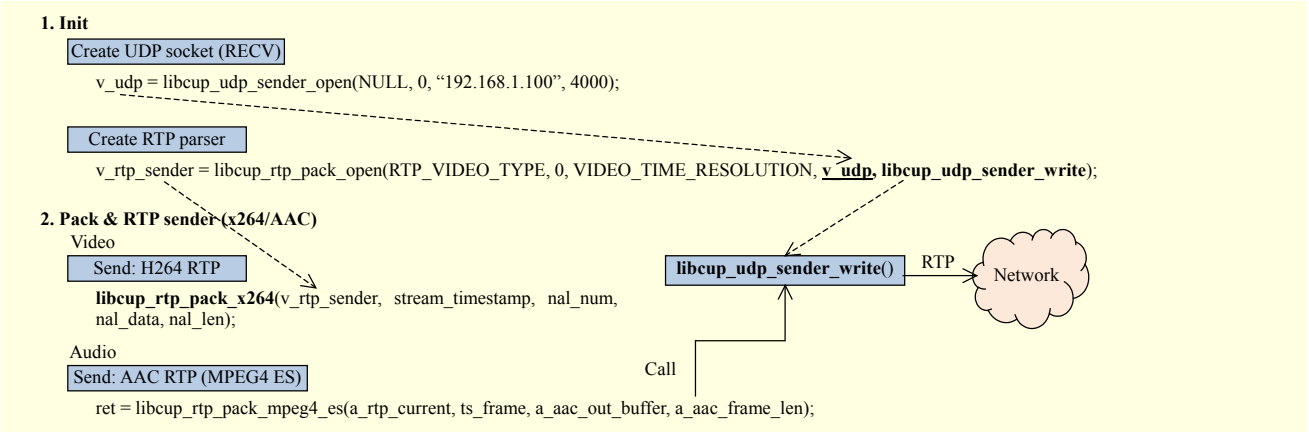


Fig. 3. RTP packetizer and sender flows.

ETRI Journal, Volume 35, Number 6, December 2013
http://dx.doi.org/10.4218/etrij.13.2013.0076

system support the H.264 decoding functionality. In addition, the client is responsible for capturing the action information of the input devices, such as a keyboard, mouse, and joystick, and sending them to the DSP through the UDP.

Figure 3 shows an RTP packetizer and sender. The flows are as follows:

- Create the UDP socket for the RTP sending using the client IP and port number.
- Create the RTP parser.
- RTP pack for x264 video frame using such parameters as rtp_handler, timestamp, number of NALs, and length.
- RTP pack for AAC audio using such parameters as rtp_handler, timestamp, and data length.
- Deliver to the client using the libcup_udp_sender_write() callback function.

IV. Implementation Results

In this section, we explain the implementation results of the proposed system for a game service platform supporting multiple users and devices based on real-time streaming.

1. Audio/Video Capturing

A. Audio Capturing

To separately capture several game sounds running on the game server, we use a virtual sound driver and common memory architecture in the shared memory method instead of the stored PCM file method previously mentioned in section III.

Figure 4 shows the virtual audio device structure for storing shared memory when using our system. In several game sounds captured separately, we modify the driver source using the /Windk/7600.16385.1/src/audio/msvad in a Microsoft WDK 7600.16385.1 sample source code [20], [21]. Most hardware drivers for PCI- and DMA-based audio devices are based on the Port Class library, which is accessible through the PortCls system driver (Portcls.sys). PortCls is an audio port-class driver that Microsoft includes as part of its operating system. PortCls supplies a set of port drivers that implement most of the generic kernel streaming filter functionality. Therefore, PortCls simplifies the task of the audio driver developer.

Typically, the port drivers provide the majority of functionality for each class of audio subdevice. For example, the WaveRT port driver does most of the work required to stream audio data to a DMA-based audio device, whereas the miniport driver provides device-specific details, such as the DMA address and device name [22]. Real Wave PCM is stored at the WaveCycle in the miniport driver. In addition, the MiniportWaveCyclicStreamMSVAD::CopyTo() function is

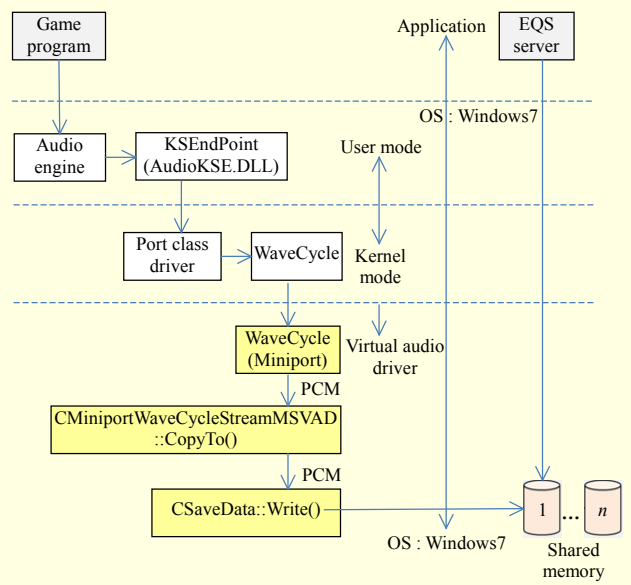


Fig. 4. Virtual audio device structure for storing shared memory.

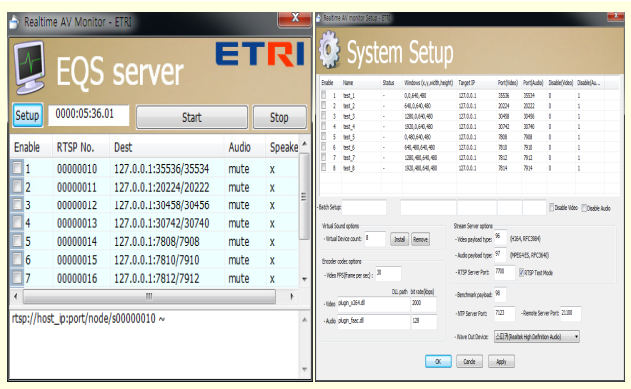


Fig. 5. Encoding server for monitoring and EQS system setup environments using the bitrates, audio/video encoder types, and virtual devices.

then used to write call the CsaveData::Write() function to the shared memory.

B. Video Capturing

The video capture module has seamless capturing functionality between the Windows API hooking windowed mode and full screen mode using the DirectDraw and Windows API hooking methods. In windowed mode, screen capturing is conducted using DirectDraw. When a game screen is switched to full screen mode, the screen capturing is processed by hooking the window event handler.

2. Audio/Video Encoding

The video codec supports the x264 and CUDA264 formats

Table 4. Encoding performance in x264 and CUDA264 for 1,000 frames.

x264/CUDA264	640×480	800×600	1024×768
Encoding time (s)	3.800/2.044	5.900/2.636	9.300/3.651
CPU usage (%)	25/17	24/17	25/15
1 frame time (ms)	3.8/2.1	5.9/2.7	9.3/3.7
Frames per second (fps)	262.7/485.3	170.5/376.3	107.6/271.7

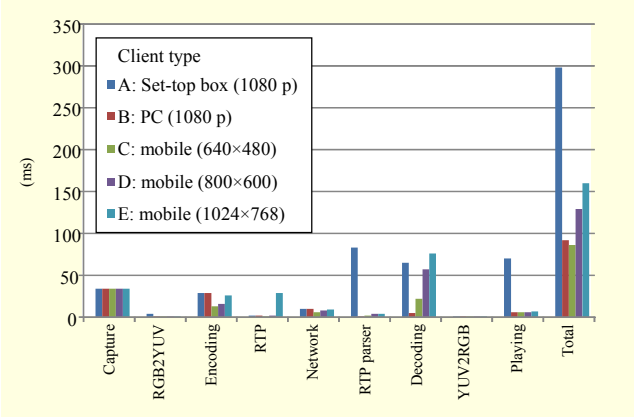


Fig. 6. Latency time tested by set-top box (1080 p) and various smart devices.

based on H.264, and the audio codec supports the AAC format, as shown in Table 3. Video codec is a parallel encoding type supporting multiple users at the same time, the number of available CPUs using video slicing for the thread to perform the encoding. In parallel, each tool is used within the MMX SIMD code by enabling the optimization of high-speed encoding.

Figure 5 shows our parallel encoding and setup EQS server used in the game server. It also shows the IP, port number, and network status of the connected client. Our video codec also supports NVIDIA CUDA264. We utilize the CUDA toolkit and SDK Tools version 4.0.17 for video encoding. NVIDIA provides high-quality GPU-accelerated libraries that developers can use for video codec operations on NVIDIA GPUs [23]. The NVIDIA CUDA Video Encoder (NVCUVENC) is a hybrid CPU/GPU accelerated library that creates a stream complaint with AVC/H.264 (MPEG-4 Part 10 AVC, ISO/IEC 14496-10). NVCUVENC takes advantage of hundreds of CUDA cores to accelerate the encoding of H.264. The inputs are YUV frames, and the outputs are generated NAL packets streaming to the client devices. This encoder supports up to High Profile Level 4.1.

We implement our encoding performance on a game server running Windows 7 OS with an Intel Core i7-2600K CPU, 4 GB of memory, and NVIDIA GTX 560Ti. There are two types of game motions, which are tested by CUDA264 and x264. The encoding performance of our system is shown in Table 4. Our encoding system can encode in 2.83 ms on average for CUDA, and 6.33 ms for x264.

Table 5. Results of latency time during beta test.

		Step			Total (ms)
		Server	Streaming	Client	
Client	Mobile	17.6	17.6	72.7	107.9
	PC/notebook	27.4	1.9	56.4	85.8
	Set-top box	29.3	50.3	57.2	146.8

We evaluate the latency time on various types of clients, that is, A (set-top box, 1080 p) and B (low-cost PC, 1080 p), and smart devices, such as C (Tablet, CPU: i7-M620, 2.66 GHz, 2Core, 640×480), D (Galaxy S2, cotex-a8, 800×600), and E (Galaxy Tab 10.1, Tegra2 cotex-a9 dual, 1024×768), as shown in Fig. 6.

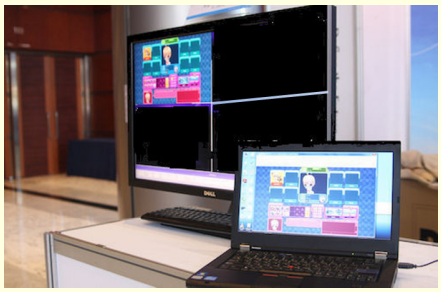
As previously mentioned, the time between a player's key action and the time the actual resulting game output appears on the player's screen, namely, the latency time or turn-around time, is the most important element for playing online games in real-time environments. In Fig. 6, most of the latency times are under 200 ms. In the case of FHD (1080 p) or higher resolution, our system needs to optimize and enhance the performance for supporting reasonable latency.

To inspect the stability of our system, our cloud-based gaming service platform is developed through beta testing with the Korea Telecom (KT) IDC Center. The cloud-based gaming service platform consists of a DSP server and two DSP nodes for mobile and tablet device users, four DSP nodes for PC users, and three DSP nodes for IPTV set-top box users. We collect data on 40 game users who use KT internet services with various devices, including 10 users for IPTV set-top boxes, 10 users for smartphone and tablet devices, and 20 users for PCs, over three months. Table 5 shows the results of latency time during the beta test.

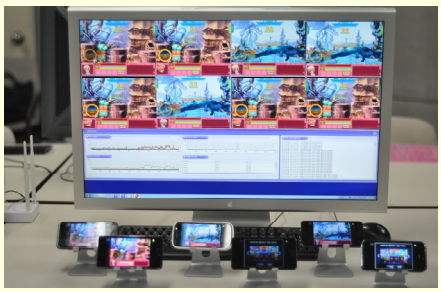
V. Conclusion

In this paper, we presented a system architecture and its use as a game service platform supporting multiple users and devices based on real-time streaming, as shown in Fig. 7. The proposed system provides real-time capturing, encoding, and streaming to the client. We showed an example of a game service platform running and managing a game screen performed on the server, and the resulting screen was provided

(a)



(b)



(c)

Fig. 7. Game service platform supporting multiple users and devices: (a) set-top box (1080 p) connected to one user, (b) tablet PC connected to four users, and (c) smart mobile devices connected to eight users.

to multiple devices, such as low-cost PCs, smart devices, and set-top boxes, as shown in Fig. 7.

There are many advanced technologies for future work. It is possible to use new capabilities and optimizations to improve the performance of video encoding, especially H.264/AVC through GPU-based implementation. We also need to focus on reducing the game server latency throughout the GPU virtualization and input/output device virtualization.

References

[1] A. Jurgelionis et al., "Platform for Distributed 3D Gaming," *Int. J. Comput. Games Technol.*, vol. 2009, Jan. 2009, pp. 231863:1-231863:15.

[2] A. Jurgelionis et al., "Testing Cross-Platform Streaming of Video Games over Wired and Wireless LANs," *Proc. IEEE 24th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Perth, Australia, Apr. 2010, pp. 1053-1058.

[3] S. Perlman, "The Process of Invention: OnLive Video Game Service," lecture, Botwinick Lab, The Fu Foundation School of Engineering and Applied Science, Columbia University, New York, NY, USA, Nov. 13, 2009. http://tv.seas.columbia.edu/videos/545/60/79

[4] "GDC 2012: Gaikai CEO David Perry Talks What Gaikai Offers," YouTube video, posted by GamerLiveTV, uploaded Mar. 22, 2012. http://www.youtube.com/watch?v=6lN5E_pgBsU

[5] Y. Tzruya et al., "Games@Large — A New Platform for Ubiquitous Gaming and Multimedia," *Proc. Broadband Europe Conf.*, Geneva, Switzerland, Dec. 11-14, 2006, pp. 11-14.

[6] I. Nave et al., "Games@Large Graphics Streaming Architecture," *Proc. 12th Annual IEEE Int. Symp. Consum. Electron.*, 2008, pp. 1-4.

[7] Ubitus, "Our Technology," Ubitus Inc., 2013. http://www.ubitus.net/en/technology.html#CC

[8] G. Humphreys et al., "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters," *ACM Trans. Graph.*, vol. 21, July 2002, pp. 693-702.

[9] S.M. Jang et al., "Client Rendering Method for Desktop Virtualization Services," *ETRI J.*, vol. 35, no. 2, Apr. 2013, pp. 348-351.

[10] L. Cheng et al., "Realtime 3D Graphics Streaming Using MPEG-4," *Proc. IEEE/ACM Workshop Broadband Wireless Serv. Appl.*, 2004, pp. 1-16.

[11] T. Karachristos, D. Apostolatos, and D. Metafas, "A Real-Time Streaming Games-on-Demand System," *Proc. 3rd Int. Conf. Dig. Interactive Media Entertainment Arts*, vol. 349, Athens, Greece, Sept. 10-12, 2008, pp. 51-56.

[12] D. de Winter et al., "A Hybrid Thin-Client Protocol for Multimedia Streaming and Interactive Gaming Applications," *Proc. 16th Int. Workshop Netw. Operating Syst. Support Dig. Audio Video*, Newport, RI, USA, Nov. 22-23, 2006, pp. 15:1-15:6.

[13] NVIDIA, NVIDIA GRID™ GRID GPUs, grid boards, NVIDIA Corporation, 2013. http://www.nvidia.com/object/grid-boards.html

[14] "GTC 2012 Keynote (Part 03): The NVIDIA Kepler GPU Architecture," YouTube video, posted by NVIDIA, uploaded May 16, 2012. http://www.youtube.com/watch?v=TxtZwW2Lf-w

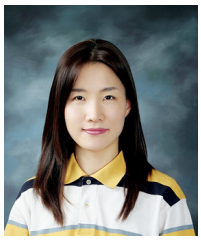
[15] S.S. Kim, K.I. Kim and J.H. Won, "Multi-view Rendering Approach for Cloud-Based Gaming Services," *3rd Int. Conf. Adv. Future Internet*, 2011, pp102-107.

[16] J. Andrews, "Designing the Framework of a Parallel Game Engine," Intel Corporation, Santa Clara, CA, USA, June 2009. http://software.intel.com/en-us/articles/designing-the-framework-of-a-parallel-game-engine

[17] ISO/IEC 14496-3, *ISO/IEC 14496-3:2001/Amd.1:2003 — Bandwidth Extension*, *ISO/IEC 14496-3:2001/Amd.2:2004 — Parametric Coding for High Quality Audio*, *ISO/IEC 14496-3:2001 and ISO/IEC 14496-3:2001/Amd.1:2003/DCOR1*.

[18] H. Schulzrinne, *RTP: A Transport Protocol for Real-Time Applications*, STD 64, RFC 3550, July 2003.
[19] S. Wenger, *RTP Payload Format for H.264 Video*, RFC 3984, Feb. 2005.
[20] Microsoft, "Microsoft Virtual Audio Device Driver Sample," WDK 8.1 sample, Microsoft Corporation, 2013. http://code.msdn.microsoft.com/windowshardware/virtual-audio-device-3d4e6150
[21] Microsoft, *Getting Started with WDM Audio Drivers*, white paper, Microsoft Corporation, Apr. 30, 2004. http://msdn.microsoft.com/en-us/library/windows/hardware/gg463452.aspx
[22] Microsoft, "Introduction to Port Class," *Audio Miniport Drivers*, design guide, Microsoft Corporation, 2013. http://msdn.microsoft.com/en-us/library/windows/hardware/ ff536829(v=vs.85).aspx
[23] NVIDIA, "NVIDIA Video Codec SDK," NVIDIA Corporation, 2013. https://developer.nvidia.com/nvidia-codec-libraries

Kyoung Ill Kim received his M.S. in computer information systems from Korea University, Jochiwon, Rep. of Korea, in 2001. Since 1983, he has worked as a principal researcher at ETRI, Daejeon, Rep. of Korea. His research interests are cloud computing, cloud game streaming, and multimedia system.



Su Young Bae received her B.S. and M.S. degrees from Kyungpook National University, Rep. of Korea, in 1998 and 2000. In April 2001, she joined ETRI, Rep. of Korea, where she is currently a senior researcher. Before joining ETRI, she was a software engineer member of the mobile communication division in Samsung Electronics. Her research interests are cloud game streaming, smart content technology, and multimedia system.

Dong Chun Lee received his M.S in computer engineering from Kyungpook National University, Rep. of Korea, in 2000. Since 2001, he has worked as a senior researcher at ETRI, Daejeon, Rep. of Korea. His research interests are game physics, GPU-based H.264 encoding, game input virtualization, and game server load test.

Chang Sik Cho received his Ph.D. from Chungnam National University, Rep. of Korea, in 2011 and his B.S. and M.S. degrees from KyungPook National University, Rep. of Korea, in 1993 and 1995, respectively. In January 1995, he joined ETRI, Rep. of Korea where he is currently a principle researcher. His research interests are cloud game streaming, smart content technology, and embedded multimedia system.

Hun Joo Lee received his BS, MS, and PhD in computer science and engineering from Chung-Ang University, Seoul, Rep. of Korea, in 1991, 1993, and 1998, respectively. In 1998, he joined ETRI, Daejeon, Rep. of Korea. He was a postdoctoral researcher at Iowa State University, Ames, IA, USA, from 2001 to 2002. His current research interests include game artificial intelligence, serious gaming, and smart content.

Kyu Chul Lee received his B.E., M.E., and Ph.D. degrees in computer engineering from Seoul National University in 1984, 1986, and 1996, respectively. In 1994, he worked as a visiting researcher at the IBM Almaden Research Center, San Jose, CA, USA. From 1995 to 1996, he worked as a visiting professor at the CASE Center at Syracuse University, Syracuse, NY, USA. He is currently a professor in the Department of Computer Engineering at Chungnam National University, Daejeon, Rep. of Korea. His current areas of interest include multimedia database system, hypermedia systems, object-oriented systems, and digital libraries. He has authored over 100 technical articles published in various journals and conferences. He is a member of ACM, the IEEE Computer Society, and the Korea Information Science Society.