J Inf Process Syst, Vol.19, No.4, pp.439~449, August 2023
https://doi.org/10.3745/JIPS.04.0280

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Resource Efficient AI Service Framework Associated with a Real-Time Object Detector

Jun-Hyuk Choi, Jeonghun Lee, and Kwang-il Hwang[*]

Abstract
This paper deals with a resource efficient artificial intelligence (AI) service architecture for multi-channel video streams. As an AI service, we consider the object detection model, which is the most representative for video applications. Since most object detection models are basically designed for a single channel video stream, the utilization of the additional resource for multi-channel video stream processing is inevitable. Therefore, we propose a resource efficient AI service framework, which can be associated with various AI service models. Our framework is designed based on the modular architecture, which consists of adaptive frame control (AFC) Manager, multiplexer (MUX), adaptive channel selector (ACS), and YOLO interface units. In order to run only a single YOLO process without regard to the number of channels, we propose a novel approach efficiently dealing with multi-channel input streams. Through the experiment, it is shown that the framework is capable of performing object detection service with minimum resource utilization even in the circumstance of multi-channel streams. In addition, each service can be guaranteed within a deadline.

Keywords
Adaptive AI Service, Multichannel Streaming, Object Detection, Real-Time, Resource Efficient

1. Introduction

Recently, thanks to the rapid development of deep learning technology, new applications and services based on real-time camera images are being actively developed. In particular, various video security systems using deep learning technologies such as object detection, facial recognition, and object tracking have made a great contribution to automating tasks such as monitoring and tracking that have been performed by humans.

Object detection technology plays the most basic role in detecting and tracking various objects in an image. Various object detection techniques based on deep learning—R-CNN, Faster R-CNN, and YOLO [1]—have dramatically improved the recognition rate and have achieved even more accurate object recognition than human vision.

Until now, the main goal of video security systems has been to record and monitor videos acquired from multiple cameras. A digital video recorder (DVR) or network video recorder (NVR) plays an important role in such video security systems. However, the existing commercial DVR and NVR systems do not have sufficient hardware resources capable of processing videos obtained from multiple channels.

Manuscript received February 1, 2023; first revision February 28, 2023; accepted March 2, 2023.
* **Corresponding Author:** Kwang-il Hwang (hkwangil@inu.ac.kr)
Dept. of Embedded Systems Engineering, Incheon National University, Incheon, Korea (jhjh1593@naver.com, junghoon450@gmail.com, hkwangil@inu.ac.kr)

In addition, most object detection models are designed not for multi-channel object detection but for a single channel image. Therefore, to handle multiple channel videos, it is necessary that the security systems include an additional artificial intelligence (AI) server equipped with more powerful hardware in parallel with DVR or NVR. Furthermore, the more channels are serviced, the more hardware resources are required.

In this paper, we propose a real-time multi-channel object detection service framework using minimal hardware resources. In the proposed method we deploy the parallel adaptive frame control (AFC) at the image input stage and use an adaptive queue which is rearranged dynamically reflecting the processing result of each input image. Thus, it enables to run multi-channel object detection task using minimum resources.

The organization of this paper is as follows. Section 2 introduces the research on object detection for multi-channel images and presents the limitations and disadvantages of existing technologies. Section 3 describes the proposed real-time remote object detection service. In Section 4, performance evaluation is conducted on several important metrics, and this paper concludes in Section 5.

2. Related Work

Among a variety of deep learning-based object detection models, YOLO is the most popular with providing convenience in use and accurate detection performance. The YOLO open-source development environment makes it easy for users to apply the YOLO model for their own application. The YOLO development team is also providing continuous version maintenance and upgrade—YOLOv2, YOLOv3, and YOLOv4 [1]. However, since the YOLO is basically designed for a single-channel object detection, it is not fit to apply for multi-channel video services as it is.

Recently, as interest in AI services and multiple video streams increases, research on those issues is being actively conducted. First, Parsola et al. [2] proposed a Hadoop system architecture to store a large number of video streams from multiple cameras and find useful information therefrom. Karimi-Mansoub et al. [3] proposed a technique for performing object detection on multiple video streams using YOLOv3 on a single graphics processing unit (GPU) through a multi-threaded structure using a multiplexer (MUX) and a demultiplexer (DEMUX). Tan et al. [4] used YOLO in parallel to track specific objects from multiple video streams. In particular, the authors proposed a technique for predicting the location information of a specific object using long short-term memory (LSTM) and tracking the object based on the information. Kim and Park [5] proposed an efficient cooperative framework between a camera device and an image analysis server. Mattela et al. [6] proposed a new enterprise class deep neural network (EcDNN) that can simultaneously perform object detection and face recognition in a video surveillance system. Aslan and İleri [7] performed benchmarking on a mobile processor for a multi-thread-based facial recognition technology. Wang et al. [8] proposed an efficient structure for an object detection pipeline on GPU using CUDA, and Liu et al. [9] proposed a parallel object tracking technique for illumination variation. In addition, GStreamer-based DeepStream developed by NVIDIA provides a dedicated software development kit (SDK) that can implement multi-channel object detection and tracking technology that can be executed on Linux and various embedded artificial intelligence processors in the form of plug-ins.

In summary, some of them have tried to modify the deep neural network structure, and some of them also tried to execute image processing tasks in a parallel structure. However, the former makes it hard to port it to other services, the latter requires more powerful hardware resources for processing multi-channel video streams in parallel. Eventually, even though there have been a lot of research on multi-channel video stream and AI services, none of existing studies could not address multi-channel video stream AI service with minimum hardware resources.

3. Adaptive Multi-Channel AI Service Framework

In this section, we present our remote AI service framework for multi-channel video streams. In the proposed framework, we employ a flexible local server that can be associated with conventional video storage systems such as DVRs and NVRs. This structure allows the framework to support various types of input video.

The framework is designed based on a modular structure. In addition, since it is designed abstracting hardware such as GPU, it can be run on various systems with only software installation without hardware-dependent settings. We also define a client as each of the video source from IP cameras or video files.

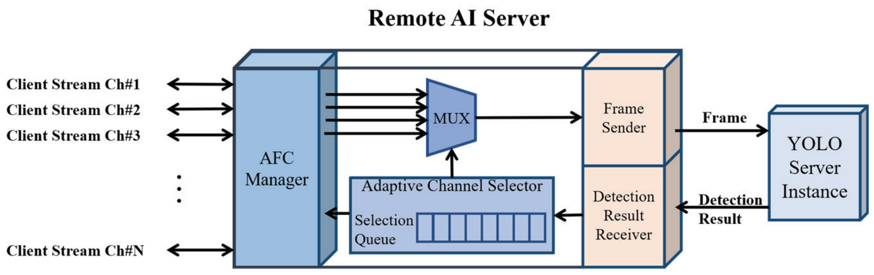

Fig. 1. Overall architecture of remote AI server.

As shown in Fig. 1, the proposed AI server is composed of AFC Manager, adaptive channel selector (ACS), Frame Sender, and Object Detection Result Receiver. Here, AFC [10] is an efficient solution to solve the cumulative delay occurring in network streaming environment, which is the inherent problem of YOLO. The original AFC is designed to be used with a single channel object detection model, but in our framework, the AFC is applied at each input stage to enable consistent processing in various input sources.

3.1 AFC Manager

In this section, the detailed structure of the AFC Manager is introduced. As shown in Fig. 2, AFC Manager is largely composed of client access manager (CAM) and multiple AFC instances. A new AFC instance is created when a client requests a service to CAM.

A client refers to an independent video channel, and CAM supports the RTSP protocol commonly used by network cameras, DVRs, and NVRs. In addition, USB cameras or local files are supported as independent channels.

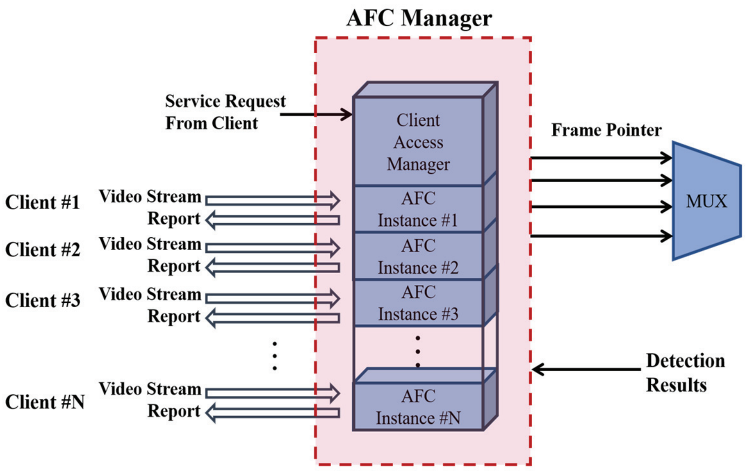

Fig. 2. Multi-channel AFC Manager.

When the CAM receives a client's service request, it goes through an authentication process that determines the type, size, frame per second (FPS), and target FPS of the image to be processed, and then creates an AFC instance after checking out whether it is available to create an AFC instance in the system. The created AFC instance operates in a multi-threading manner and consists of a Frame Receiver, Frame Scaler, and AFC. Frame Receiver receives video streams to be processed and stores it in memory. The Frame Scaler is used to rescale the original image size to the size appropriate to be input to the YOLO server instance. In our framework, AFC is utilized to synchronize the FPS of the input image with the target FPS based on the object detection result (Fig. 3).

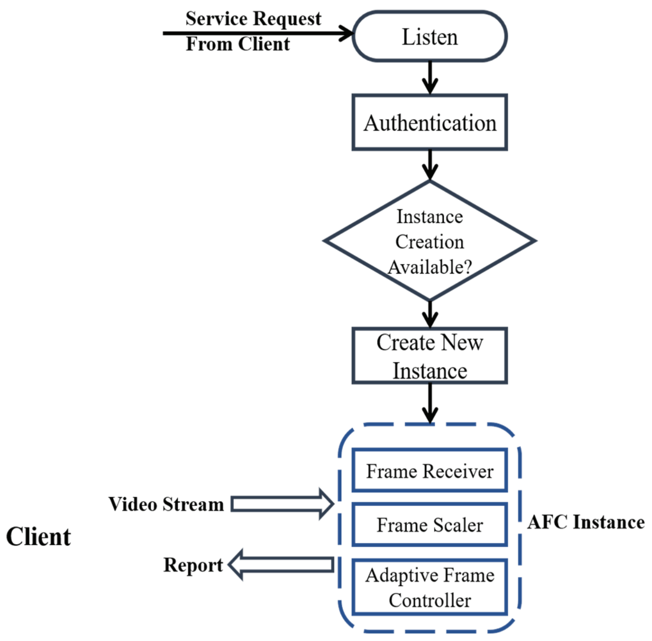

Fig. 3. Service flow of client access manager.

3.2 Adaptive Channel Selector

In this section we present the ACS module that is to transmit selective frames to YOLO instance server. The overall structure is shown in Fig. 4. The inputs of the MUX are frame pointers of all client channels to be processed, and the output of the MUX is determined by the channel index of the ACS.

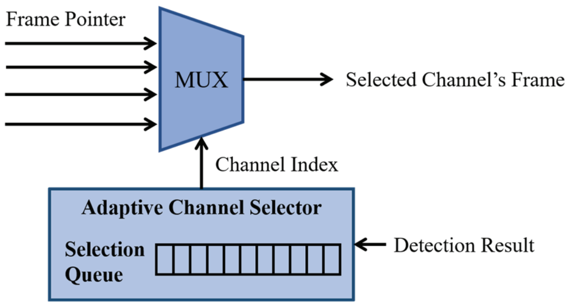

Fig. 4. Adaptive channel selector.

The output of the MUX also goes to the Frame Sender unit to deliver it to YOLO Server instance. Since each client's frame information is handled by a pointer, it is more efficient that each frame is transferred as a pointer, rather than each frame data content is transferred to the Frame Sender. Afterwards, the object detection result returned by YOLO server instance is used as an input to the ACS and the channel selection queue is reallocated based on the result. Eventually, the result of the channel selection queue reallocated is reflected on the MUX by indicating next channel number.

Fig. 5 shows how channel index is determined by channel selection queue. Tolerable delay of an application is defined as T_d. N is the number of video channels requesting the service. When the remote AI server is first run, the initial length of the channel selection queue is first determined, and channel indexes are stored in the queue in order. As soon as the queue creation is completed, MUX outputs each frame indicated by the channel index in order from the head of the queue. The frame is delivered to YOLO server instance and then the object detection result is feedbacked to ACS. Whenever ACS receives an object detection result, it measures c, the current processing FPS of YOLO server instance.

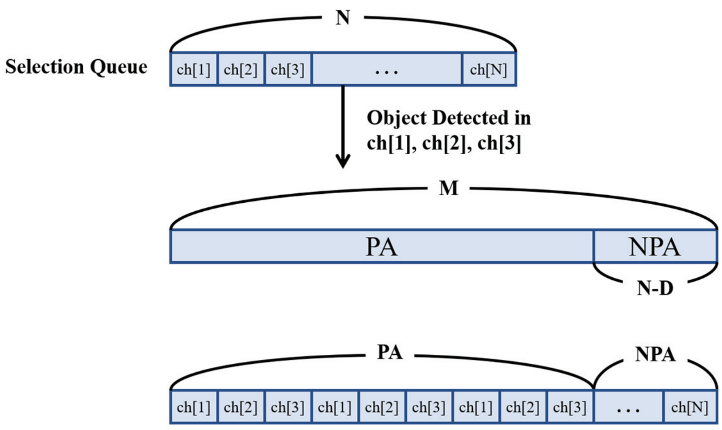
N
Selection Queue
ch[1]
ch[2]
ch[3]
...
ch[N]
Object Detected in ch[1], ch[2], ch[3]
M
PA
NPA
N-D
PA
NPA
ch[1] ch[2] ch[3] ch[1] ch[2] ch[3] ch[1] ch[2] ch[3] ... ch[N]

Fig. 5. Channel assignment from multi-channel stream.

In order to efficiently manage the channel selection queue, ACS dynamically varies the queue size according to recent object detection results. Initial queue length is equal to the number of requesting channels, N. However, after any object in some channels is detected, the queue is rearranged dividing into two area, the privileged area (PA) and non-privileged area (NPA). The former contains channel indexes having objects detected in the previous round. And the latter contains channel indexes having no object in the previous round. Here, a round is defined as a period when the service of all the channel in the queue is completed once. The reason that the queue is divided into two section is for the channels occurring frequently objects detected to have more opportunity (slots) to be serviced by YOLO server, by expanding PA area. If an object is detected in at least a channel, ACS starts to allocate slots in the queue as follows:

- First, it expands the maximum queue length by M (Eq. 3.1)
- Next, it calculates available PA size (Eq. 3.2)
- Next, it allocates evenly slots for the channels in which objects are detected.
- If there are empty slots remaining in the PA area, channels with high object detection probability are assigned to the area in the order of preference.
- Lastly, it allocates channels not detected onto NPA, $N - D$.

$$M = \mathrm{c} \times T_d, \tag{1}$$

$$length(PA) = M - (N - D). \tag{2}$$

3.3 YOLO Server Instance

This section describes a communication method between remote AI server and YOLO service instance. In order to minimize computational load of YOLO service instance, additional computations such as image size conversion are not performed in YOLO instance. Instead, when remote AI server converts the size of the frame according to the appropriate neural network input size in advance. Communication between YOLO service instance and remote AI server is made concisely as shown in Fig. 6. All the communication used for sending frames and returning results are based on TCP.

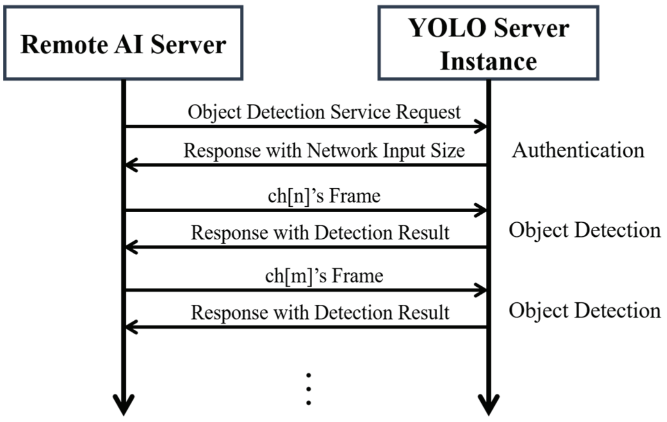
Remote AI Server
YOLO Server Instance
Object Detection Service Request
Response with Network Input Size
Authentication
ch[n]'s Frame
Response with Detection Result
Object Detection
ch[m]'s Frame
Response with Detection Result
Object Detection

Fig. 6. Communication between AI server and YOLO instance.

As shown in Fig. 6, remote AI server sends an object detection service request to the YOLO server instance. After authenticating the service request from each client, YOLO instance starts the service by returning the input size of the deep neural network. After that, YOLO server instance waits to receive the next frame to be serviced by the neural network. YOLO instance passively services the object detection of a frame depending on the ACS processing. As soon as a frame is received, YOLO server instance performs object detection and responds the results to ACS with the information such as the number of detected objects, bounding box coordinates in the image of detected objects, and detection possibility.

4. Experimental Results

4.1 Experimental Environment and Implementation Results

We implemented our remote AI server framework and tested on the various environments. Table 1 shows the basic experimental environment for our framework. In our experiment, both of YOLOv3 and YOLOv4 are tested as object detection service models. However, the two different version of models showed almost similar results. This is because our framework is operated without regard to the performance of the service models. The size of YOLO neural network input image is 416×416. The video resolution used for input video is 640×480, and frames per second is 30.

Key goal of our framework is to provide object detection service of multi-channel video in real-time with minimum overhead. Fig. 7 shows a part of our implementation result. As a result, we achieved object detection service of 12 concurrent video streams even on the laptop equipped with low-end GPU.

Table 1. Basic environment

	Specification
OS	Ubuntu 18.04 LTS
Development tools	C, C++, OpenCV (4.1.1)
CUDA	Version 10.2
CuDNN	Version 8.0.0
Hardware	Laptop with CPU(i7-8750H), Memory (16 GB), and GPU (GTX 1060 6GB)
Object detection model	YOLO (YOLOv4 416×416)
Input video	Video files (H.264, 640×480, 30 FPS)

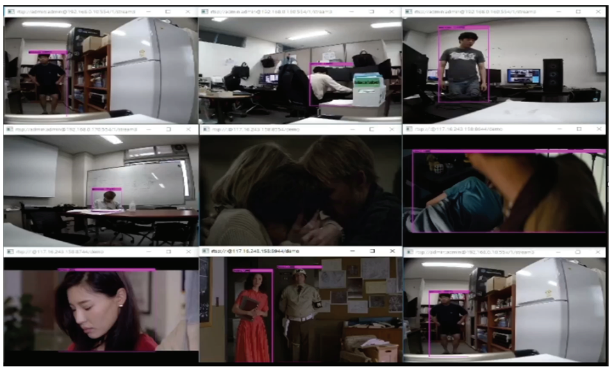
Fig. 7. Remote AI server with YOLOv4 (9 channels example).

4.2 Performance Evaluation

In order to evaluate the performance of the proposed framework, we employed two metrics: service throughput and total service delay. For comparison with our framework, we also implemented a parallel YOLO instance model, which is a conventional model for multi-channel video processing. The model is used to compare performance with our framework in each experiment.

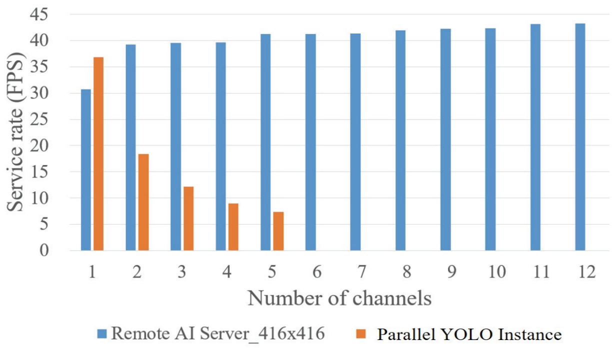

Fig. 8. Service rate.

The first experiment is to measure the service throughput of YOLO server instance according to the number of video channels. The experimental result is shown in Fig. 8. As shown in Fig. 8, in the “parallel YOLO instance” environment, as the number of video channels requesting service increases, multiple YOLO instances are simultaneously executed in one system, and thus each instance uses computing resources divided as many as the number of channels. As a result, the average throughput rapidly decreases, and YOLO instances more than six channels were not executed due to hardware restrictions (limit of VRAM memory of GPU). On the other hand, it is remarkable that the remoted AI server maintains high throughput (service rate) even in the circumstance of 12 concurrent video streams. The conventional YOLO reads an image, performs object detection, and outputs the processed frame. Therefore, it is inefficient because the process of reading, processing, and outputting the image should be performed every frame. However, the remote AI server creates an AFC instance for each client who requests the service and reads the input video frame in a separate thread according to the target FPS. So, only a single YOLO process can use GPU resource even if the number of clients (channels) increases.

The second experiment is total service delay time for each video channel. Since one of the key goals of remote AI server is to perform object detection in real time for video streams received through multiple channels, for our experiment, we assumed that the allowable deadline is 1.5 seconds. The service delay is measured by the difference between the frame sequence of the received images and the time actually processed for the input images in real time. The experimental result of 4 channel video service in the parallel YOLO environment is shown in Fig. 9. And the experimental result of our framework with 4 channels and 12 channels, respectively, is shown in Fig. 10. As shown in Fig. 9, as time goes by, the number of unprocessed and accumulated frames for the input image increases, resulting in an increase in total service delay time. From the 60th frame, the service time starts to exceed the deadline. On the other hand, as shown in Fig. 10(a), the remote AI server using an AFC instance for each channel shows that the total service delay time for all channels maintains about 0.17 seconds. In addition, as shown in Fig. 10(b), the remote AI server framework is capable of guaranteeing the real-time processing even in the circumstance of 12 concurrent video streams.

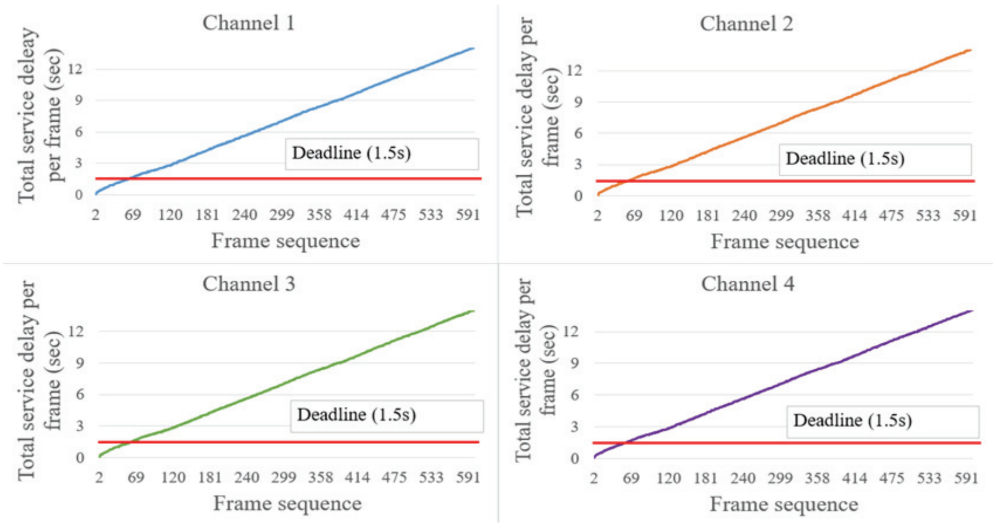

Fig. 9. Total service delay (parallel YOLO instance) of 4 channels.

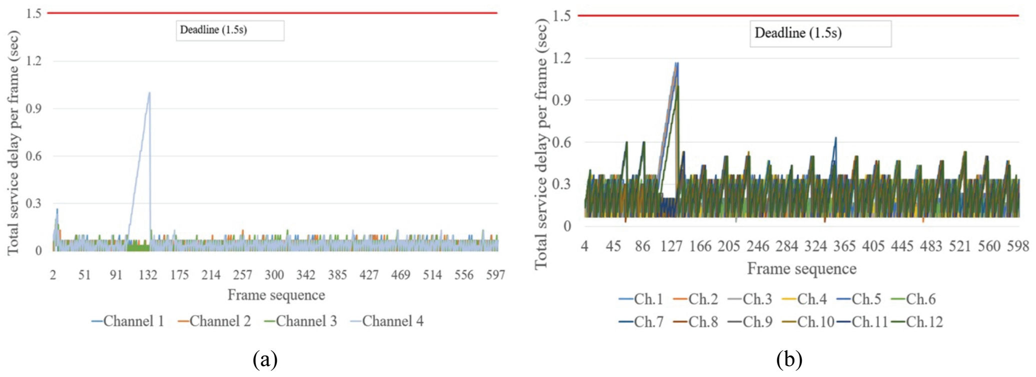

Fig. 10. Total service delay (ACS): (a) 4 channels and (b) 12 channels.

5. Conclusion

In this paper, we proposed a remote AI server associated with an object detection model to deal with multiple video streams in real-time. Our server framework is designed based on modular architecture and it can be associated with various AI models. In particular, it has been proven that our framework which has the combination of AFC-based modular instance creation and a single YOLO server instance designed to maximize GPU resource utilization can process more than 12 concurrent video streams in real time within 1.5 seconds deadline even on the laptop environment equipped with a GTX1060 GPU. Since the remote AI server were designed minimizing dependency on the underlying neural network architecture, it is possible that the remote AI server is associated with other AI service models as well as object detection. Furthermore, it is expected that it can be utilized as a core technology for AI application server to efficiently deal with multi-channel video streams.

Acknowledgement

This work was supported by Incheon National University Research Grant 2019.

References

[1] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, "Yolov4: optimal speed and accuracy of object detection," 2020 [Online]. Available: https://arxiv.org/abs/2004.10934.

[2] J. Parsola, D. Gangodkar, and A. Mittal, "Post event investigation of multi-stream video data utilizing Hadoop cluster," *International Journal of Electrical & Computer Engineering*, vol. 8, no. 6, pp. 5089-5097, 2018. https://doi.org/10.11591/ijece.v8i6.pp5089-5097

[3] S. Karimi-Mansoub, R. Abri, and A. Yarici, "Concurrent real-time object detection on multiple live streams using optimization CPU and GPU resources in YOLOv3," in *Proceedings of the 4th International Conference on Advances in Signal, Image and Video Processing*, Athens, Greece, 2019, pp. 23-28.

[4] L. Tan, X. Dong, Y. Ma, and C. Yu, "A multiple object tracking algorithm based on YOLO detection," in *Proceedings of 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Beijing, China, 2018, pp. 1-5. https://doi.org/10.1109/CISP-BMEI.2018.8633009

[5] D. Kim and S. Park, "An intelligent collaboration framework between edge camera and video analysis system," in *Proceedings of 2018 International Conference on Electronics, Information, and Communication (ICEIC)*, Honolulu, HI, 2018, pp. 1-3. https://doi.org/10.23919/ELINFOCOM.2018.8330653

[6] G. Mattela, C. Pal, M. Tripathi, R. Gavval, and A. Acharyya, "Enterprise class deep neural network architecture for recognizing objects and faces for surveillance systems," in *Proceedings of 2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, Bengaluru, India, 2019, pp. 607-612. https://doi.org/10.1109/COMSNETS.2019.8711399

[7] S. Aslan and S. C. İleri, "Performance analysis of ARM big.LITTLE architecture based mobile processor with multi-thread face detection," in *Proceedings of 2019 4th International Conference on Computer Science and Engineering (UBMK)*, Samsun, Turkey, 2019, pp. 336-339. https://doi.org/10.1109/UBMK.2019.8907058

[8] X. Wang, "An efficient end-to-end object detection pipeline on GPU using CUDA," master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2019.

[9] S. Liu, G. Liu, and H. Zhou, "A robust parallel object tracking method for illumination variations," *Mobile Networks and Applications*, vol. 24, pp. 5-17, 2019. https://doi.org/10.1007/s11036-018-1134-8

[10] J. Lee and K. I. Hwang, "YOLO with adaptive frame control for real-time object detection applications," *Multimedia Tools and Applications*, vol. 81, pp. 36375-36396, 2022. https://doi.org/10.1007/s11042-021-11480-0

Jun-Hyuk Choi **https://orcid.org/0000-0002-0406-9051**

He received B.S. degree in embedded system engineering from Incheon National University, Incheon, Korea, 2021. Since Sep 2020, he is with the embedded system engineering from Incheon National University as a master's course. His current research interests include embedded system, image processing system.

Jeonghun Lee **https://orcid.org/0000-0002-9617-2901**

He received B.S. degree in embedded system engineering from Incheon National University, Incheon, Korea, 2020. Since March 2020, he is with the embedded system engineering from Incheon National University as a master's course. His current research interests include embedded system, image processing system.

Kwang-il Hwang **https://orcid.org/0000-0002-4196-4725**

He received a Ph.D. degree from the Department of Mathematics, Korea University, in 1994. From 1989 to 1999, he was a senior researcher with the Electronics and Telecommunications Research Institute, South Korea. He has been running the Digital Forensic Research Center, Korea University, since 2008. He is currently the president of the Division of Information Security, Korea University. He has authored or coauthored over 130 articles in various archival journals and conference proceedings and over 200 articles in domestic journals. His research interests include digital forensics, data processing, forensic framework, incident response, etc.