EPiC Series in Computing

Volume 58, 2019, Pages 368–377

Proceedings of 34th International Conference on Computers and Their Applications

Construction Scheme of a Scalable Distributed Stream Processing Infrastructure Using Ray and Apache Kafka

Kasumi Kato[1], Atsuko Takefusa[2], Hidemoto Nakada[3], and Masato Oguchi[4]

[1] Ochanomizu University,
2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112-8610, Japan
kasumi@ogl.is.ocha.ac.jp

[2] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
takefusa@nii.ac.jp

[3] National Institute of Advanced Industrial Science and Technology (AIST),
2-3-26 Aomi, Koto-ku, Tokyo, 135-0064, Japan
hide-nakada@aist.go.jp

[4] Ochanomizu University,
2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112-8610, Japan
oguchi@is.ocha.ac.jp

Abstract

The spread of various sensors and the development of cloud computing technologies enable the accumulation and use of large numbers of live logs in ordinary homes. To operate a service that utilizes sensor data, it is difficult to install servers and storage in ordinary homes and to analyze the collected data from sensors. Those data are typically transmitted from sensors to a cloud and analyzed in the cloud. However, services that involve moving image analysis must transfer large amounts of data continuously and require high computing power for analysis. Hence, it is highly difficult to process them in real time in the cloud using a conventional stream data processing framework. In this research, we propose a construction scheme for a highly efficient distributed stream processing infrastructure that enables scalable processing of moving image recognition tasks according to the amount of data that are transmitted from sensors. We implement a prototype system of the proposed distributed stream processing infrastructure using Ray and Apache Kafka, which is a distributed messaging system, and we evaluate its performance. The experimental results demonstrate that the proposed distributed stream processing infrastructure is highly scalable.

1 Introduction

The spread of various sensors and the development of cloud computing technologies enable the accumulation and use of many live logs in ordinary homes. These technologies are applied to various services such as watching the elderly and children and crime prevention measures.

In addition, deep learning technology has been widely used for image and speech recognition processing. Deep learning is a machine learning method that uses multilayered intermediate layers that perform identification in a neural network. Many deep learning libraries, such as TensorFlow [1] and Chainer [2], have been developed. However, a key issue for deep learning is heavy processing loads.

To operate a service that utilizes sensor data, it is difficult to install servers and storage in ordinary homes and analyze the collected data from sensors. Typically, those data are transmitted from sensors to a cloud and analyzed in the cloud. However, services that involve moving image analysis require large amounts of data to be transferred continuously and high computing power for the analysis; hence, it is difficult to process them in real time in the cloud using a conventional data processing framework.

We propose a construction scheme for a highly efficient distributed stream processing infrastructure that enables scalable processing in moving image recognition according to the amount of data that is transmitted from sensors.

First, we perform preliminary experiments using Apache Spark [3] (hereinafter called Spark) and Ray [4]. Spark is a representative cluster computing platform that is designed to be fast and versatile, and Ray is a newer distributed execution framework. We investigate the characteristics of their distributed recognition processing and demonstrate that Ray enables scalable distributed processing. However, according to our previous work [6], it is difficult to perform effective distributed processing using Spark because it aims at high-throughput processing of multiple user tasks based on a data affinity policy. In the preliminary experiments, the image identification processes are distributed and parallelized using Spark and Ray with various parameters, and the behaviors of each distributed process are visualized. Image data that are identified are provided by the ImageNet [5]. We demonstrate that scalable distributed processing can be performed using Ray from the experimental results.

Next, we implement a prototype system of the proposed distributed stream processing infrastructure using Ray and Apache Kafka [7] (hereinafter called Kafka), which is a distributed messaging system, and demonstrate its performance. In the previous work [9], we investigated the performance characteristics of Kafka and confirmed that a Kafka cluster with an appropriate configuration enables the performances of large amounts of image data messaging. Therefore, we employ Kafka as a messaging system for the prototype system. In the experiments on the prototype system, Kafka transmits image data to the Ray cluster and we investigate the performance of distributed recognition processing using PyTorch [8] and TensorFlow, and neural network libraries. The experimental results show that the implemented distributed stream processing infrastructure that uses Ray and Kafka is highly scalable.

2 Distributed stream processing infrastructure

In this research, we assume a large-scale distributed stream data processing infrastructure, as shown in Fig. 1. When moving image data are transmitted from the sensors and cameras that are installed in each home to the cloud, the moving image data are processed by the stream processing infrastructure and passed to the distributed processing platform. When the distributed processing platform receives the data, the data analysis processing is performed by the deep learning framework and the result is returned to the service management system. The stream processing infrastructure is an infrastructure that can continuously process an infinitely long stream of data. By using the stream processing infrastructure, it is possible to process data continuously and in real time. In this system, it is assumed that image data that are acquired from sensors in each home are processed as a stream. The distributed processing infrastructure

Table 1: Performance of the computer used in the experiment

OS	Ubuntu 16.04LTS
CPU	Intel(R) Xeon(R) CPU W5590 @3.33 GHz 4 core×2 sockets(8 core)
GPU	NVIDIA GeForce GTX 980
Memory	48 Gbyte

is the foundation on which a large amount of data is efficiently processed using clusters that are composed of many computers. In this system, identification processing of a large amount of image data is performed in a cluster and the image data are identified in a short time.

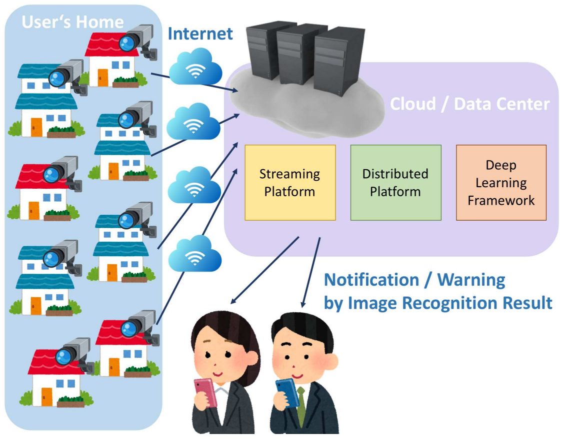

Figure 1: Assumed infrastructure for large-scale distributed stream processing.

Specifically, we propose the distributed stream processing infrastructure that is shown in Fig. 2. Since it was confirmed that Kafka can construct a scalable messaging system, we use Kafka as a stream processing infrastructure. Data are transferred from the Kafka cluster to the Spark cluster or the Ray cluster and image identification processing is performed by their workers in the cluster. The Kafka Producers are placed in the Clients and the image data that are acquired from the sensors are sent to the Broker in the Kafka Cluster. The Broker transmits the acquired image data to the Kafka Consumer running on the worker nodes of the Spark or Ray cluster and the image data recognition processing is performed in the Consumers.

3 Distributed processing efficiencies in Spark and Ray

We investigate the distributed processing efficiency of Spark and Ray, as preliminary experiments. We construct clusters of Spark and Ray and aim at high efficiency of image identification processing using PyTorch and TensorFlow as the back end of PyTorch. In the experiments, ImageNet is used as a data set. For each case of Spark and Ray, we measure the time that it takes for images to be evaluated by each worker after execution of the program and for the result to be returned to the master. We prepare 10000 ImageNet data files, each of size 26 KB, for the

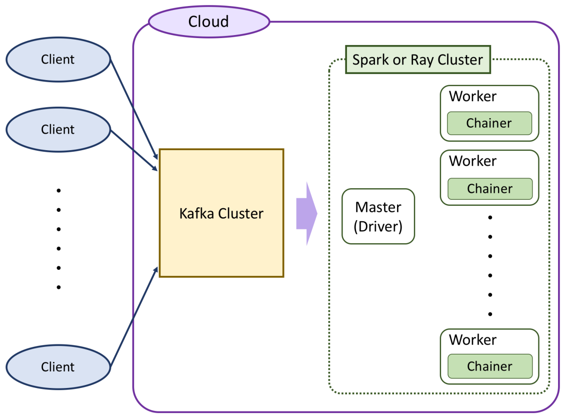

Figure 2: Overview of the proposed distributed stream processing infrastructure, which uses Kafka and Spark or Ray.

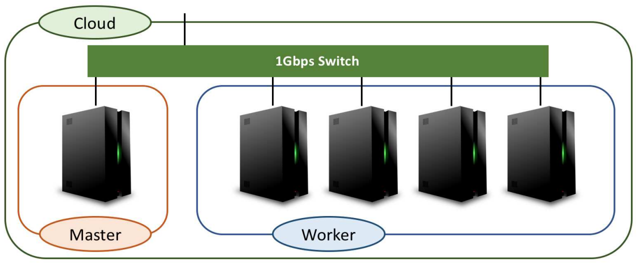

Figure 3: Experimental environment.

experiments. Table 1 shows the performance of a cluster node used in the experiments. Nodes with the same performance are used for the master and all workers. Each node is connected by a 1 Gbps network as shown in Fig. 3. In the experiments, we use Spark v. 2.3.1, Ray v. 0.5.3, PyTorch v. 0.4.1, and TensorFlow v.1.8.0.

3.1 The efficiency of Spark distributed processing

Fig. 4 shows master-worker processing in the case of Spark. The round rectangle drawn by a solid line represents the entire Spark cluster and the dotted rectangles represent a physical node. Among the physical nodes, a node that is used as a master is represented by a red dotted line and a node that is used as a worker is represented by a blue dotted line. Distributed processing in Spark is performed as follows: (1) Execute the Python program on the master. (2) Make Spark read the ImageNet data and create the RDD. (3) Pass the created RDD to the worker. (4) Identify ImageNet data using PyTorch in workers. RDD is a fault tolerance distributed data set, and Spark automatically distributes data by using RDD and its processing methods provided by Spark. Each node is connected in Spark Standalone Mode.

The results when the number of nodes is changed from 1 to 5 and the number of partitions is changed from 8 to 48 in 8 increments, are shown in Fig. 5. The number of partitions is a

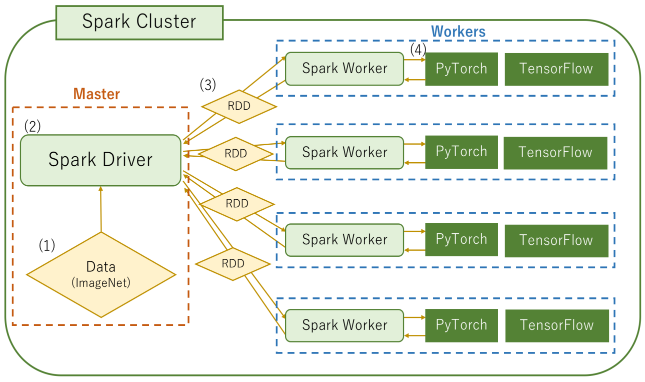

Figure 4: Master-Worker processing using Spark and PyTorch

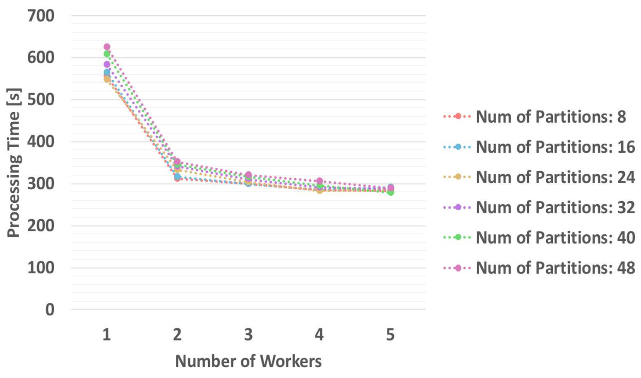

Figure 5: Processing times of distributed processing by Spark.

parameter that determines how many tasks are to be divided and distributed to workers. In general, the larger the number of partitions, the smaller the task bias between workers becomes. In Fig. 5, the horizontal axis indicates the number of nodes and the vertical axis indicates elapsed time; each color shows different experimental results when the number of partitions is set from 8 to 48. According to Fig. 5, the processing time is reduced as the number of nodes increases, while the results obtained by different numbers of partitions are almost comparable.

3.2 The efficiency of Ray distributed processing

Fig. 6 shows master-worker processing in the case of Ray. The round rectangle drawn by a solid line represents the entire Ray cluster and the dotted rectangles represent physical nodes. Among the physical nodes, a node that is used as a master is represented as a red dotted line and a node that is used as a worker is represented as a blue dotted line.

Distributed processing in Ray is performed as follows: (1) When a Python program is executed on the master, (2) the Ray driver in the master node places the data in the object store of its own node and contacts the local scheduler. (3) The local scheduler communicates

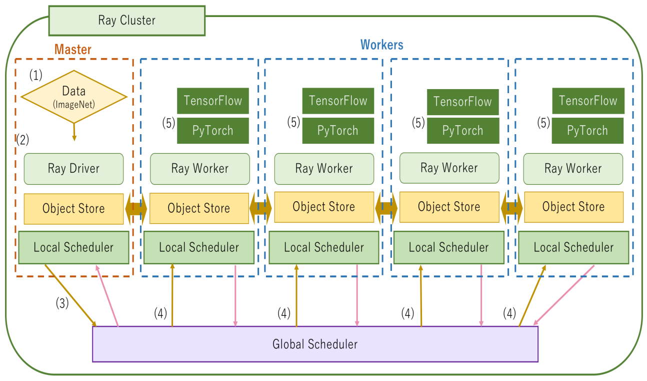

Figure 6: Master-Worker processing using Ray and PyTorch.

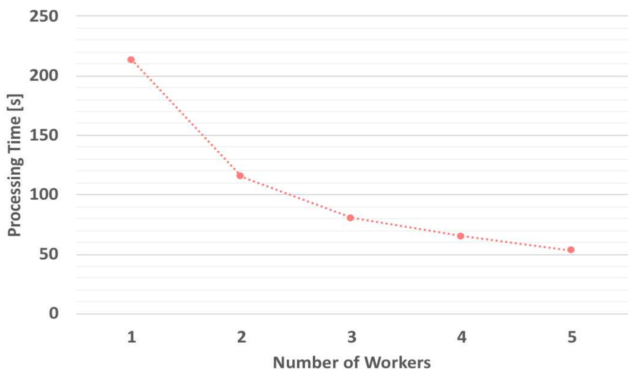

Figure 7: Processing times of distributed processing by Ray.

to the global scheduler. (4) The global scheduler sends instructions to local schedulers of each node. (5) Ray workers evaluate ImageNet data using PyTorch. The data are stored and shared in Object Store deployed over the Ray cluster nodes.

As with Spark, we measure the execution times of Ray on 10000 tasks as we change the number of nodes from 1 to 5. Since Ray does not have the concept of a partition as Spark does, we distribute data to workers 1 to 5 in a round-robin manner. The average values of 10 measurements is shown in Fig. 7. The horizontal axis is the number of nodes and the vertical axis indicates elapsed time. In Fig. 7, as the number of nodes is increased, the processing time is reduced. It seems that Ray's ImageNet data identify process is highly scalable.

Table 2 shows comparison of the elapsed times of Spark and Ray. The results of phase (1) show the duration from the start of the program to the start of the identification process. The phase (2) results indicate the time to identify 10000 ImageNet data. The configuration of Spark process is that number of workers is 1, 5 and the number of partitions is 40. The configuration of Ray is that number of workers is 1, 5. According to table 2, the times for identification comparable between Spark and Ray. However, it seems that Spark takes time for the phase (1) due to resource management, task scheduling or another reasons. Therefore, Ray can process

Table 2: Comparison of identify processing time, spark and ray.

# of workers	1		5	
Phase	(1)	(2)	(1)	(2)
Spark	392.873	213.682	229.235	49.897
Ray	1.686	226.133	7.123	48.890

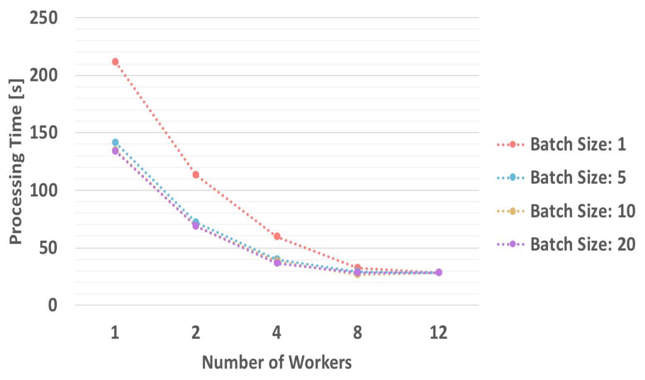

Figure 8: Processing times of batch-based distributed processing by Ray.

faster than Spark.

Further, in this experimental configuration, batch processing can be applied for remote functions. Batch processing is a method of identifying multiple image data simultaneously together after storing a specified amount of data. It is possible to convert many matrix×vector executions to a few matrix×matrix executions, thereby possibly shortening the overall processing time. The number of worker nodes is varied among 1, 2, 4, 8, 12; and the batch size is varied among 1, 5, 10, 20. Fig. 8 shows the average values of 10 measurements of processing 10000 images. The horizontal axis is the number of workers and the vertical axis indicates the elapsed time; Fig. 8 is drawn for the results of different batch sizes. In the Fig. 8, increasing the batch size from 1 to 5 can shorten the processing time by 70 seconds when the number of workers is 1. However, even if the number of workers and the batch size are increased, the processing time converges to approximately 28 seconds.

4 Distributed stream processing infrastructure

According to the investigation of the distributed processing efficiencies in Section 3, Ray is faster and more scalable than Spark for the distributed parallel identification processing of an image.

Fig. 9 shows the proposed distributed stream processing infrastructure using Ray and Kafka. Kafka Broker receives image data from Kafka Producers in the clients and passes them to the Kafka Consumers that are deployed in the workers in the Ray cluster. Each Consumer performs identification processing. Again, experiments are conducted using image data from ImageNet. The computers used in the experiments have the same performance as those used in Section 3. In the experiments, we use Kafka v. 1.1.0.

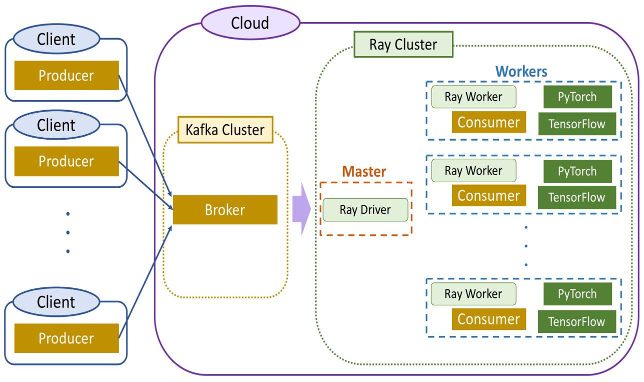

Figure 9: Distributed stream processing infrastructure using Ray and Kafka.

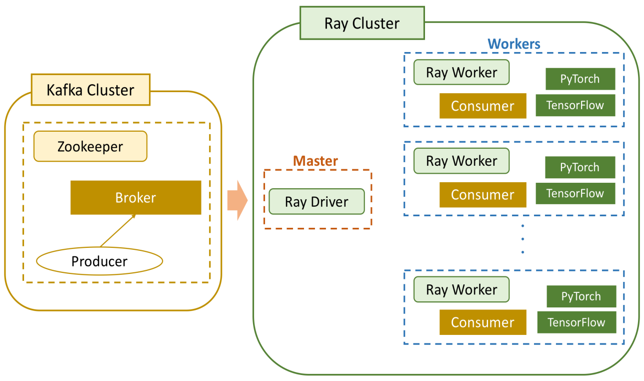

Figure 10: Distributed stream processing composition by Ray and Kafka.

4.1 Adaptation to a streaming environment by Kafka

We perform distributed stream processing using Kafka and Ray. The experimental configuration is shown in Fig. 10. The round rectangles drawn by solid lines represent the Ray cluster and the Kafka cluster and the dotted rectangles represent physical node. When Kafka is started and a Python program is executed, image data are transmitted from Producer to Broker. Upon receiving the data, the Broker sends the data to the Consumers deployed in the worker nodes in the Ray cluster and the Consumers identify the image using PyTorch and TensorFlow.

The number of Producers is set to 1, the number of Ray workers is varied among 1, 2, 4, 8, 12, and the batch size is varied among 1, 5, 10, 20. The experimental results are shown in Fig. 11. The horizontal axis is the number of worker nodes and the vertical axis is the throughput; a graph is drawn for each batch size. Here, the throughput is calculated from the number of image data processed per second. Fig. 11 shows that the throughputs are improved by approximately a factor of two compared with batch sizes of 1 and 5, while the throughputs of batch size 10 and 20 are comparable. In addition, the throughputs of all batch sizes are improved as the number of workers increases. Thus, the experimental results show that a scalable distributed stream processing infrastructure can be constructed using Kafka and Ray.

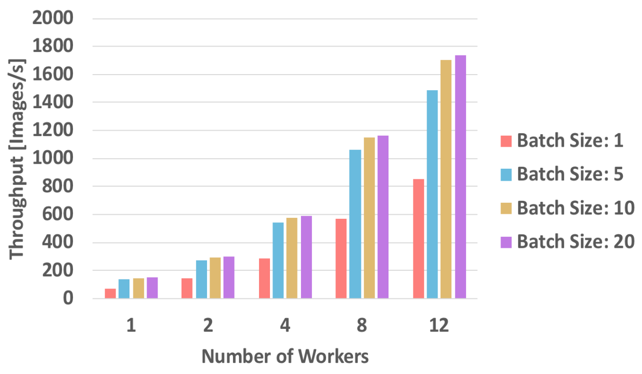

Figure 11: Throughputs of batch-based distributed stream processing by Kafka and Ray.

5 Related Work

Chen et al. [10] use the Producer-Consumer model of Kafka as a streaming data acquisition layer for moving images and the real-time processing framework Spark Streaming, combined with OpenCV [11], as a data processing layer, with memory, HDFS and HBase [12] used as data storage layers and Web Technology for final result display. The performance is evaluated by adjusting the numbers of worker nodes and batch slices and the CPU usage rate. This method can extract simple information from a video. Further development requires the detection of moving objects and the recognition of behaviors, for which models that support CNN and 3D CNN are necessary. Our research uses TensorFlow, which is supporting CNN deep learning framework.

In our research, we aim at accelerating distributed stream processing using a general data processing infrastructure.

6 Conclusions

We proposed a construction scheme of a distributed stream processing infrastructure using Kafka and Ray. In the preliminary experiments, image processing by PyTorch was distributed in parallel, using Spark or Ray and those efficiencies were investigated. We found that scalable distributed processing is possible using Ray. Next, we implemented a distributed stream processing infrastructure using Ray and Kafka. The experimental results showed that it is possible to implement a highly scalable distributed stream processing infrastructure with Ray and Kafka. Moreover, its performance is further improved by batch processing. In the future, we will consider adapting the system to the processing of moving images, increasing the sizes of Kafka and Ray clusters, and conducting experiments in a larger-scale environment to improve the performance.

Acknowledgments

This paper is partially based on results that were obtained from a project that was commissioned by the New Energy and Industrial Technology Development Organization (NEDO), JSPS KAKENHI Grant Number JP16K00177, and the open collaborative research at National Institute of Informatics (NII) Japan (FY2018).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, http://download.tensorflow.org/paper/whitepaper2015.pdf. pp. 1- 19. [Online]. http://tensorflow.org/.

[2] S. Tokui, K. Oono, S. Hido, and J. Clayton, “Chainer: a next-generation open source framework for deep learning,” in In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), 2015, 6 pages.

[3] “Apache Spark,” https://spark.apache.org/.

[4] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liax, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, “Ray: A Distributed Framework for Emerging AI Applications,” 2017. http://ray.readthedocs.io/en/latest/index.html/.

[5] J. Deng, W. Dong, R. Socher, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database.” IEEE Conference on Computer Vision and Pattern Recognition, 2009. http://www.image-net.org/.

[6] K. Kato, A. Takefusa, H. Nakada, and M. Oguchi, “Consideration of parallel data processing over an apache spark cluster,” Poster Session in IEEE International Conference on Big Data (2017).

[7] “Apache kafka,” https://kafka.apache.org/.

[8] A. Paszke, S. Gross, S. Chintala, and G. Chanan, “Pytorh: Tensors and dynamic neural networks in python with strong gpu acceleration,” 2017. Available: https://pytorch.org/.

[9] A. Ichinose, A. Takefusa, H. Nakada, and M. Oguchi, “A Study of a Video Analysis Framework Using Kafka and Spark Streaming,” Second Workshop on Real-time and Stream Analytics in Big Data (2017).

[10] H. Chen, F. Luo, L. Zhao, and Y. Li, “Design and Implementation of Real-Time Video Big Data Platform Based on Spark Streaming,” International Conference on Computer Science and Application Engineering (CSAE), 2017.

[11] “OpenCV,” https://opencv.org/

[12] “HBase,” https://hbase.apache.org/