A Study of Load Balancing between Sensors and the Cloud for a Real-Time Video Streaming Analysis Application Framework

Yuko Kurosaki Ochanomizu University
2–1–1 Otsuka
Tokyo, Japan
yuko-k@ogl.is.ocha.ac.jp

Atsuko Takefusa National Institute of Advanced Industrial Science and Technology (AIST)
1–1–1 Umezono
Ibaraki, Japan
atsuko.takefusa@aist.go.jp

Hidemoto Nakada National Institute of Advanced Industrial Science and Technology (AIST)
1–1–1 Umezono
Ibaraki, Japan
hide-nakada@aist.go.jp

Masato Oguchi
Ochanomizu University
2–1–1 Otsuka
Tokyo, Japan
oguchi@is.ocha.ac.jp

ABSTRACT

For the home and office, many life-log analysis applications for transferring data from cameras and sensors to the cloud and analyzing the data have been developed. However, because of limitation of the resources of the cloud and the network bandwidth between the sensor and the cloud, it is difficult to execute large load processing, such as video streaming analysis, in real time on the cloud. Moreover, taking into account the execution environment from the sensor to the cloud, it is necessary to set an appropriate degree of parallelism in the processing from the pre-processing, such as feature extraction, to the analysis on the information. In this paper, we propose a video streaming analysis application framework for load balancing between sensors and the cloud, and investigate the performance of the application in a cluster environment that simulates the sensor and the cloud. From the experiments, we show that the processing performance is improved by increasing the number of processing threads, and we demonstrate the effectiveness of load balancing between the sensors and the cloud.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

IMCOM '16, January 04-06, 2016, Danang, Viet Nam
© 2016 ACM. ISBN 978-1-4503-4142-4/16/01...$15.00
DOI: http://dx.doi.org/10.1145/2857546.2857601

Keywords

Load Balancing, Apache Storm, Stream Data, Feature Extraction, On-line Machine Learning, Jubatus

1. INTRODUCTION

To obtain data from cameras and sensors at home and the office, such as for the purpose of safety services for children and the elderly and security, many life-log analysis applications have been developed. In the case of using the applications at home, processing on the cloud is needed because installing servers, storage and analysis at home are difficult. However when transferring a large amount of sensor data, including video data, to the cloud, processing in real time is not possible because of the limitation of the resources of the cloud and because of the network bandwidth between the sensors and the cloud. Moreover, for large-capacity stream data, such as video streaming, it is necessary to perform high-load processing continuously from feature extraction to analysis on the information.

As opposed to the method of performing all processing on the cloud, the paradigm called Fat Client, which provides server functions to the server on the sensor side (client side) and reduces the load of processing on the cloud, is useful. In addition, edge computing, which is aimed at reducing the response delay of the services and the load of the cloud by linking a small edge server located near the sensors or users [1], and Fog Computing, which prevents overconcentration to the cloud by placing a distributed processing environment between the cloud and devices [2], are proposed. To perform video streaming analysis in real time, taking into account the execution environment from sensors to the cloud, it is necessary to set an appropriate degree of parallelism for each process.

In this paper, we develop a video streaming analysis application framework for load balancing between sensors and the cloud, and investigate the performance of the application in a cluster environment that simulates the sensor and cloud. This framework is developed using Apache Storm (hereinafter called Storm), which provides a distributed stream computing foundation and is able to perform the high-load processing of video streaming analysis. In evaluation experiments of the performance considering changes of the number of Bolt threads and load balancing between sensors and the cloud considering the network bandwidth, we demonstrate the effectiveness of load balancing between sensors and the cloud. The average processing time of a tuple does not change, and the processing performance is improved by increasing the number of processing threads, although the interval time from receiving a tuple to receiving the next tuple increase. In the case of the low-bandwidth environment, the total number of images that can be processed for the elapsed time is increased by load balancing between sensors and the cloud, and the distributed processing that links fat client and the cloud is effective.

This paper is organized as follows: Section 2 describes the related technology. Section 3 explains the proposed video streaming analysis application framework. Section 4 evaluates the framework. Section 5 introduces related work. Finally, we conclude in Section 6.

2. RELATED TECHNOLOGY

The video streaming analysis application uses Storm as the basis of load balancing between the sensors and cloud and uses on-line machine learning Jubatus for analysis. We provide a summary of each software below.

2.1 Apache Storm

Storm is a distributed, real-time computing system developed by Back Type [3]. Storm has been acquired by Twitter Inc. and is used for real-time analysis of tweets on Twitter. Apache Hadoop performs batch processing on a large scale, whereas Storm specializes in real-time distributed processing. Apache Spark[4] is in the same stream processing framework. Spark, which performs batch processing, executes stream processing as a chain of batch processing for events received for 1 second. Therefore, the throughput of Spark is improved, but the processing of Spark is slower than pure stream processing because of an approximately 0.5 to 2.0 second delay [5].

Storm deploys Topology to Storm clusters and performs processing. Topology is a network structure that consists of Spout and Bolt, as shown in Figure 1. Spout is a starting point to start a flowing stream, and Bolt is the process that converts the flowing data. Stream is a set of tuples that is continuously sent, and it is possible to use a standard data type and a user-defined type for added serialization code as Stream.

In addition, Storm has two types of modes: Local mode and Distributed mode. Local mode can execute Storm clusters on a single computer. Distributed mode dynamically loads and executes balanced processes that are defined in Topology to multiple nodes on a Storm cluster. We use the Distributed mode in our study. In Distributed mode, Nimbus manages the entire Storm cluster, and Supervisor manages the individual Slave nodes. Zookeeper coordinates the Storm cluster. We send Topology to Nimbus with a command, and Workers, which are started by Supervisor, execute the processing of Spout or Bolt. The number of Workers is defined when starting Supervisor, and threads

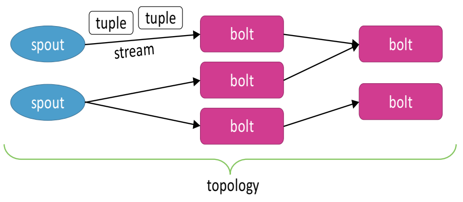


Figure 1. An example of the Storm topology.

designated in the definition in Topology are distributed to the Workers.

2.2 On-line Machine Learning: Jubatus

Jubatus was developed by Preferred Infrastructure and NTT SIC, and it is an on-line machine learning framework[6]. The elements ”stream (on-line) processing”, ”parallel distributed processing” and ”deep analysis” have a trade-off relationship, but Jubatus meets the requirements of these elements. For batch machine learning, servers synchronize every time a model is updated. The update frequency is low, so it does not significantly affect performance. On the other hand, for the distributed processing of on-line machine learning, the update frequency of the learning model increases and the synchronization cost increases. Therefore, the server processing performance deteriorates, and a framework satisfying the three elements at the same time has not been developed. Usually, UPDATE processing shares data between servers and updates the learning model. However, in Jubatus, the data themselves are not shared in UPDATE, only the analysis model after the learning of each server is shared in MIX. The process has to allow for real-time and large-scale parallel and distributed processing.

Jubatus uses Datum, which is a key-value in analysis processing. Datum has three types of values: string data, number data, and binary data. It is possible to transform multimedia data, such as images and sounds, into binary data. From these data, the data conversion module of Jubatus extracts features that are required in machine learning processing. The feature vector converter of Jubatus can customize the feature extraction process in a JSON file, and we can use a plug-in for the feature extractor. The plug-in is a dynamic library file (.so), and it is used by opening a path with a JSON file.

3. VIDEO STREAMING ANALYSIS APPLICATION FRAMEWORK

In this section, we describe the design overview of the video streaming analysis application framework that we propose in this study. Video streaming analysis has three essential steps: (1) getting image data, (2) feature extraction, and (3) machine learning. To obtain and process all sensor data on the cloud side, after process (1) on the sensor side, the sensor data, including a moving image, are sent to the cloud. Servers on the cloud side perform processes (2) and (3). However, for load balancing between the sensors and cloud, part of process (2) is distributed to servers on the sensor side to reduce the amount of data transfer between the sensor and cloud servers and to reduce the load of the cloud servers.

When performing all processing on the cloud side, traffic

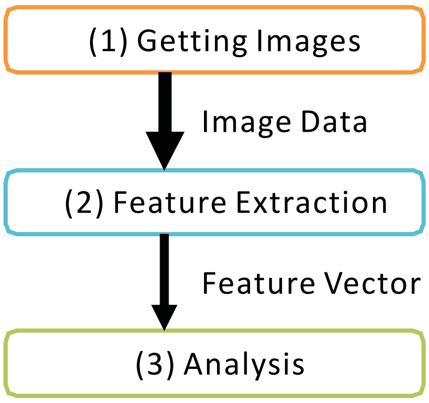


Figure 2. Flow of video streaming analysis application framework.

to the cloud is bigger, so performance degradation caused by the overhead is an issue. When performing processing on both the sensor side and the cloud side for (2), the amount of traffic to the cloud is reduced, but the load of feature extraction on the sensor side becomes large.

3.1 Design of Video Streaming Analysis Application Framework

The implemented application is divided into the following three processes shown in figure 2.

1. Getting Image Data from WEB Camera
2. Feature Extraction using Bag-of-Features
3. Image Classification using Jubatus

We describe the details of each process below.

3.1.1 Getting Image Data from WEB Camera

First, we implement part of getting image data from a WEB camera on the sensor side, which is the origin of the data stream. Using OpenCV [7], image data are acquired by a WEB camera and are stored in the structure. The structure is transferred to (2) Feature Extraction using the Bag-of-Features.

3.1.2 Feature Extraction using Bag-of-Features

The next steps are feature extraction and vectorization of the image data using OpenCV. The vectorized data are transferred to (3) Analysis in Jubatus.

Bag-of-Features, based on the dictionary data clustering from the set of local features made from an image to the features using the K-means method, is a method that uses a histogram of the number of feature points with the feature values that belong to each group.

Bag-of-Features extraction in our study is performed as follows. First, we extract the local features from an image using OpenCV. Local features have several types. SIFT, the most common of the types, is a robust feature value that includes constant rotation and illumination changes, the scaling. Although the recognition accuracy is a little bit worse than for SIFT, SURF performs lighter and faster processing of the feature point detection. In this study, we use the SURF feature values as local features. SURF extracts some of the characteristic points in the image, called the keypoints. Each keypoint consists of the feature vector of 128 dimensions, but it is difficult to use a feature vector of the entire image for machine learning because the number of extracted keypoints varies depending on the image. Therefore, we cluster the local features and create a dictionary by considering the center vector of each cluster as the characteristic pattern called a Visual Word. We match the feature vector group extracted from the image to the Visual Word using this dictionary and represent the frequency characteristic pattern features of the entire image using a histogram. Once the dictionary is created, the number of Visual Words is constant, so we can convert the feature amount of each image into vector data of the same size.

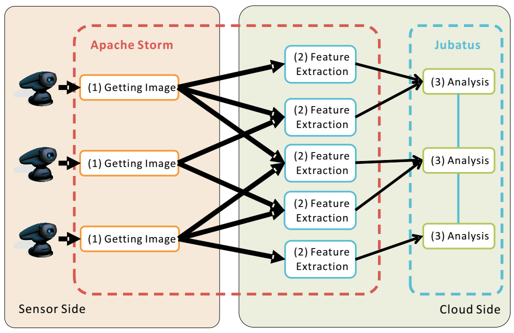
Next, we convert the generated vector data to a format that is processed in Jubatus. As described in Section 2.2, Jubatus uses the data format called datum. Thus, we convert the generated histogram to datum. We list the format of each value of the histogram, store it in datum and then convert the values into data that can be transferred to Jubatus. Finally, the system establishes a connection with the Jubatus server and transfers the datum storing the feature vector acquired from the image to the Jubatus server.

3.1.3 Image Classification using Jubatus

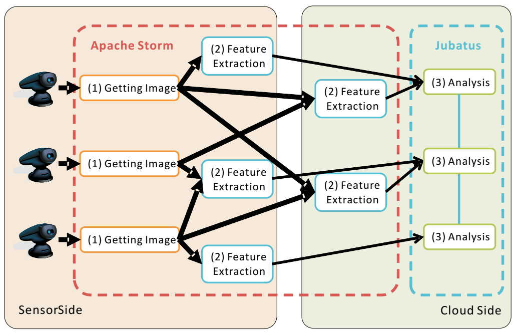
Various analyses, such as classification, recommendations, and linear regression, are possible in Jubatus. We perform classification learning using Classifier API. By using the trainAPI before video streaming analysis, Jubatus performs supervised learning in advance. To classify using the learning result, we use classifyAPI. Datum that is sent to the Jubatus server is analyzed through the two-step data conversion of filtering and feature extraction. The filter step removes information that is unnecessary for learning, and the feature extraction step extracts features from the filtered data. What elements are removed from the data in the filter processing, which algorithm is used and how the feature extraction is weighted are set by a JSON file that is specified on server startup. We use a default filter and a given value as the weight in the feature extraction. We select the Adaptive Regularization of Weight vectors (AROW) as the algorithm for classification and set the sensitivity parameters for learning to 1.0. The result from the Jubatus server is sent to a user.

3.2 Implementation of the Framework using Storm

We use Storm because the video streaming analysis application can perform real-time processing more rapidly by executing the heavy processing in parallel as necessary. The processing for (1) that is described in Section 3.1 is implemented as Spout processing, and the processing for (2) and (3) is implemented as Bolt processing. Distributed processing using the sensor and cloud is achieved by constructing a Storm cluster that contains servers on the sensor and cloud sides. In the image data analysis application used in this study, the data traffic between (1) and (2) is large, and the load of processing (2) is relatively high. Therefore, as shown in Figure 3, the processing of (1) is executed on the sensor side, the processing of (2) is executed on the sensor and/or



Execution on cloud side



Execution on sensor and cloud side

Figure 3. Configurations of the framework.

cloud side, and the processing of (3) is executed on the cloud side.

4. EVALUATION EXPERIMENTS

In this study, we perform the following two experiments to investigate the performance in the case of load balancing between sensors and the cloud side of feature extraction in the video streaming application framework.

exp1) Evaluation experiments of the performance considering changes in the number of Bolt threads

exp2) Evaluation experiments of the load balancing between sensors and the cloud considering the network bandwidth

Each experiment compares the number of jobs completed per elapsed time. exp1) also compares the average processing time per stream datum with the change in the number of processing threads and the tuple receive interval on the Bolt thread.

4.1 Overview of experiments

In the experiments, we discriminate two human behaviors using the implemented system. First, Jubatus learns two human behaviors by using 100 images of 320×240 pixels: "sit on the chair" and "open the door". By selecting image data randomly each time defined in Spout, we extract the features of the selected image data. The number of Visual Words in the feature extraction is set to 100. In exp1), as shown in Table 1, the number of Spout threads is set to two, the inter-arrival time of image data is set to 10 ms/tuple, and the number of Bolt threads is 8, 16, 24, or 32. In exp2), as shown in Table 2, the number of Spout threads is set to two, the number of Bolt threads is set to 16, the inter-arrival time of image data is set to 10 ms/tuple, and the network bandwidth between sensors and cloud varies 10, 50, 100, and 1000 Mbps. We use PSPacer [9] for network bandwidth control.

Table 1. Measurement parameters (exp1)

Number of Spout Threads	2
Number of Bolt Threads	8, 16, 24, 32
Inter-arrival Time of Image Data	10[ms/tuple]
Network Bandwidth	1000[Mbps]

Table 2. Measurement parameters (exp2)

Number of Spout Threads	2
Number of Bolt Threads	16
Inter-arrival Time of Image Data	10[ms/tuple]
Network Bandwidth	10, 50, 100, 1000[Mbps]

The constructed Storm cluster is shown in Figure 4. Both the computer on the sensor side and the computer on the cloud side use an Intel Xeon W5590 ((3.33 GHz, 4 core)×2 socket). Preparing five Supervisor nodes, we set one of the Supervisor nodes on the sensor side and the others on the cloud side computer. Each Supervisor node has 8 workers, which is the number of cores. One of the Supervisor nodes on the cloud side also acts as a Nimbus node. We use a computer running Zookeeper on the cloud side.

4.2 Experimental Results

4.2.1 *Evaluation experiments of performance considering changes of number of Bolt threads*

We evaluate the performance by considering the change of the number of Bolt threads. Figure 5 shows the result of the number of jobs processed for the elapsed time. The vertical axis represents the total number of processed jobs, and the horizontal axis represents the time in seconds. As the number of Bolt threads increases, the number of jobs processed for the elapsed time increases. However, the number of processed jobs does not increase significantly after the number of Bolt threads reaches 16 because the number of Bolt threads reaches the number that can process the amount of generated data. When using an excessive number of Bolt threads for the amount of stream data, performance deterioration is not observed.

Figure 6 shows the interval time required to receive the tuple on the Bolt thread and the average processing time per 1 tuple. The vertical axis represents the interval time to receive the tuple on the Bolt thread and the average processing time per 1 tuple in milliseconds. The horizontal axis represents the number of Bolt threads. The processing time does not change when increasing the number of Bolt threads, but the interval time from receiving a tuple to receiving the next tuple increases because increasing the number of Bolt threads requires more time for the assignment from the Spout thread to the Bolt thread.

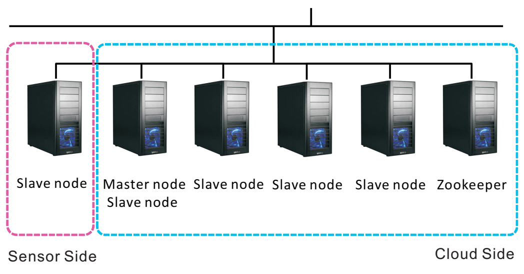


Figure 4. Experimental environment.

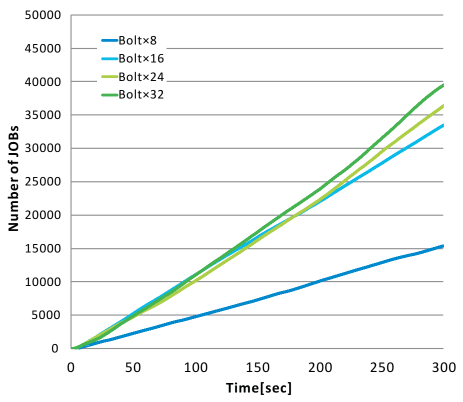


Figure 5. The number of jobs processed for the elapsed time.

4.2.2 *Evaluation experiments of load balancing between sensors and the cloud considering the network bandwidth*

We evaluate the performance of load balancing between the sensors and cloud considering the network bandwidth. In this experiment, the number of Bolt threads is 16 because the number of processed jobs does not increase significantly after the number of Bolt threads reaches 16 for 10 ms/tuple, as shown in Figure 5. Figure 7 and Figure 8 show the measurement result. The vertical axis represents the total number of processed jobs, and the horizontal axis represents the time in seconds. Table 3 shows the arrangement of the processing threads. Comparing the results when the bandwidth between the sensors and cloud is 1 Gbps and 100 Mbps shows that limiting the bandwidth causes a decrease of the number of processed jobs. When using load balancing between the sensors and cloud, the performance in the presence or absence of distributed processing between the sensors and cloud does not change at 1 Gbps. On the other hand, the number of processed jobs is increased by distributed processing between the sensors and cloud at 100 Mbps, 50 Mbps, and 10 Mbps. When the server on sensor side has rich resources (bolt×5), the number of processed jobs increases more than the case that the server has poor resources (bolt×2). Thus, when the network bandwidth is low, load distribution processing between the sensors and

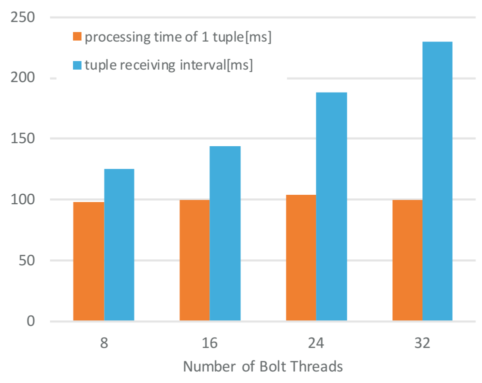


Figure 6. The interval time to receive tuples on each Bolt thread and average processing time per one tuple.

Table 3. The arrangement of processing threads in the case of distributed processing between sensors and the cloud

Number of Bolt Threads in Sensor Side	Slave Node (Sensor side)	Slave Node1 (Cloud side)	Slave Node2 (Cloud side)
0	Spout×2, Bolt×0	Bolt×8	Bolt×8
2	Spout×2, Bolt×2	Bolt×7	Bolt×7
5	Spout×2, Bolt×5	Bolt×5	Bolt×6

cloud is effective.

5. RELATED WORKS

Many video streaming analyses using cloud technology have been studied in recent years, but most of the methods focus on the efficient contents search of video data on the cloud and load balancing in the on-demand system of the video data using a cloud platform. A built-in system [10] that detects, tracks, and analyzes video streaming and an automatic monitoring system [11] that detects activities and tracks abnormal behavior of people have been developed, but these run on 1 node and are not considered scalable.

Abdullah et al. constructed a video analysis framework that obtains, processes, and analyzes video data in the cloud [12]. They proposed a scalable framework on the cloud and improved image processing speed by using a GPU. The current study is similar in terms of processing a large amount of video data using a design that is considered scalable. This method performs all processes on the cloud, but our study use the fat client model.

6. CONCLUSIONS

In this study, to achieve the real-time processing of the video streaming analysis application, considering the traffic and processing of the load, we improve the speed of preprocessing of video streaming analysis using load balancing between the sensors and cloud. We introduce Storm and design and implement video streaming analysis application framework load balancing processing between sensors and the cloud. In a large cluster environment, we perform exper-

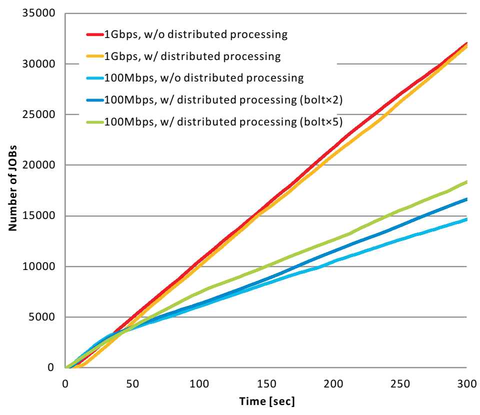


Figure 7. Comparison between the cloud processing case (w/o distributed processing) and sensors and cloud processing case (w/ distributed processing). The network bandwidths between sensors and cloud are 100 Mbps and 1 Gbps.

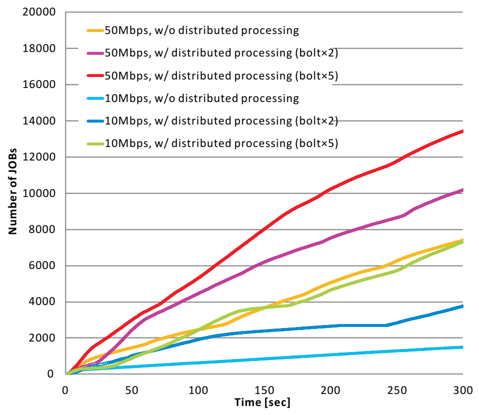


Figure 8. Comparison between the cloud processing case (w/o distributed processing) and sensors and cloud processing case (w/ distributed processing). The network bandwidths between sensors and cloud are 10 Mbps and 50 Mbps.

iments in consideration of the network bandwidth between the sensors and cloud and demonstrate the effectiveness of load balancing between the sensors and cloud.

In this study, we have experimented with constant inter-arrival time image data. However, in the real environment, the amount of streaming data changes because the framework analyzes video streaming upon motion detection. In the future, we will develop a distributed processing scheme to correspond to streaming data that vary greatly.

7. ACKNOWLEDGMENT

This paper is partially based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

8. REFERENCES

[1] Milan Patel, Brian Naughton, Caroline Chan, Nurit Sprecher, Sadayuki Abeta, Adrian Neal, "Mobile-edge computing," ETSI, Tech. Rep., 09 2014.

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli, "Fog computing and its role in the internet of things," in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ser. MCC '12. ACM, 2012, pp. 13–16.

[3] Apache Storm, https://storm.apache.org/

[4] Apache Spark, https://spark.apache.org/

[5] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica, "Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing," Electrical Engineering and Computer Sciences, UCB/EECS-2012-259, 2012.

[6] On-line Machine Learning Jubatus, http://jubat.us/ja/

[7] Image Library OpenCV, http://opencv.org/

[8] Tomoyuki Nagahashi, Hironobu Fujiyoshi, "Object Category Recognition by Bag-of-Features using Co-occurrence Representation by Foreground and Background Information," Machine Vision Applications, pp.413, 2011.

[9] Ryousei Takano, Tomohiro Kudoh, Yuetsu Kodama and Fumihiro Okazaki, "High-resolution Timer-based Packet Pacing Mechanism on the Linux Operating System," IEICE Transactions on Communications, Vol.E94.B, No.8, pp.2199-2207, 2011.

[10] B. S. System, "IVA 5.60 intelligent video analysis," Bosh Security System, Tech. Rep., 2014.

[11] Project BESAFE, http://imagelab.ing.unimore.it/besafe/

[12] Tariq Abdullah, M Fahim Tariq, Yusuf Baltaci and Nick Antonopoulos," Traffic Monitoring Using Video Analytics in CloudsTraffic Monitoring Using Video Analytics in Clouds," 7th International Conference on Utility and Cloud Computing, pp.39-48, 2014.