A Scheme to Improve Stream Data Analysis Frequency for Real-time IoT Applications

Chaxiong Yukonhiatou[1] Tomoki Yoshihisa[2] Tomoya Kawakami[3] Yoshimasa Ishi[2]
Yuuichi Teranishi[4,2] Shinji Shimojo[2]

Abstract: Due to the recent prevalence of IoT (Internet of Things) devices, stream data such as video data or sensor data are collected and analyzed for real-time applications. The frequency of analysis (analysis frequency) is one of the main factors to improve performances of some applications. For instance, the probability to identify a person in real-time can increase by frequently analyzing images got from surveillance cameras. However, communication capacities between IoT devices and processing servers limit the numbers of data to be collected in real-time and suppress analysis frequencies. To break this limitation, we propose an efficient data collection scheme with progressive quality improvement approach. In our proposed scheme, each data source produces some data of those qualities are lower than original data such as low resolution image data. Only the cases where higher quality data are needed for analyses, processing servers progressively collect them. Thus, our proposed scheme reduces average data amount of collection and data collectors can collect a larger numbers of data in real-time. We measure the analysis frequency of our proposed scheme in our developed simulator and confirm that our proposed scheme can improve the frequency.

1. Introduction

Recently, various IoT devices such as cameras and weather sensors connect to the Internet. These things are now attracting a lot of attention. These things generate stream data such as video data or temperature data and act as stream data sources. In most IoT applications, processing servers collect stream data from these IoT devices continuously and analyze them in real-time. The frequency of analysis (analysis frequency) is one of the main factors to improve performance of some applications. For example, suppose the case where a processing server receives video data continuously from surveillance cameras and analyzes image data of each frame to identify recorded people. Number of the people identified increases by frequently analyzing image data, since they are moving and the probability to record them in the video increases. Moreover, the probability to record the person can increase by collecting video data from more surveillance cameras. Hence, the number of stream data sources to be collected in real-time is also a factor to improve application performances.

A more frequent data analysis by processing servers causes a larger data amount of collection. However, communication capacities between IoT devices and processing servers are limited. Therefore, a more frequent data analysis causes a longer communication time in cases when stream data sources are stored on buffers due to the network congestions. Thus, the processing delay of data (from the time of data generation to the time to finish analyzing the data) increases. In cases of insufficient capacities of communication buffers, a larger data amount of collection causes data loss since the collected data cannot be stored to the buffers. To reduce the processing delay, many schemes have been proposed ([5], [7], [8]). These schemes degrade qualities of data, such as resolutions for image data, to reduce data to be collected and achieve a shorter delay. Quality degradations result in performance degradations of IoT applications. Applications can improve their performance by reducing data to be collected with a higher analysis frequency or a large number of stream data sources.

In this paper, we propose an efficient data collection scheme with progressive quality improvement approach. In our proposed scheme, each stream data source produces some data of those qualities are lower than original data such as low resolution image data. Only the cases where higher quality data are needed for analyses, processing servers progressively collect them (additional explanation in subsection 1). Thus, our proposed scheme reduces average data amount of collection and data collectors can collect a larger numbers of data in real-time.

The rest of this paper is organized as follows. In Section 2, we introduce some work that are related to our proposed scheme. In Section 3, we explain our assumed system environments. Our proposed scheme is explained in Section 4, and evaluated in Section 5. Finally, we will conclude the paper in Section 6.

2. Related Work

Various methods to faster analyze stream data have been proposed. A two layer system architecture for stream data analysis is proposed in [1]. In the first layer, the system executes pre-analyses to received data and determines whether proceed to main

[1] Graduate School of Information Science and Technology, Osaka University
[2] Cybermedia Center, Osaka University
[3] Nara Institute of Science and Technology
[4] National Institute of Information and Communications Technology

analysis executions in the second layer. The proposed system architecture can reduce processing loads since the system does not execute redundant main processes. Though the method divides processes, data are not divided into some parts.

A two-level indexing structure for data collection is proposed in [3]. In this method, the data are first stored to the memory having tree structures in the first level and then each data segment passes to the second level (storage) with their reference key tree. It can faster for data collection due to each data segment can store separately. Their method is different from our proposed approach in the point that they reduce data loads in the processing computer but we reduce processing delays for data analysis.

In [4], queuing models for processing stream data are analyzed for improving the processing delays and a queuing method is proposed. In the method, received stream data are stored to the buffer of processing computers. Processing computers use different buffers for each application. By considering queuing situations of other buffers, the method reduces the processing delays. In our proposed method, we can adopt this method in processing computers. Our proposed method is different from this in the point that we reduce processing delays by managing how to process data.

In [6], A dynamic bitrate adaptation scheme is proposed. In this proposed method, the approriate bitrate is selected by the system. By selecting the bitrate, the buffer is occupied to hold the bitrate and send to the processing computer. In the processing computer, the data is divided into series of segments then send to the requested users. This method is similar to our proposed approach in case of dividing the data into smaller. So. in this method we can adopt to use in the data sources. Some points are different from our proposed approach is that their method can auto adjust the bitrate according to the requested users. For example, if user use a smartphone requests for the bitrate in the server, that user will receive the different bitrate that requested by using a computer. In our proposed approach, the data quality will be requested when it is required only.

An efficient CPU resource allocation scheme for stream data analysis is proposed in [9]. By allocating CPU resources to each data stream and processing received stream data in a single scheduled execution, the scheme can enable faster stream data analysis. The approach of the scheme is effective CPU resource allocations and is different from our approach. In our approach, stream data analysis frequencies are improved by effective communications.

The method proposed in [10], can reduce the bandwidth consumption and the amount of transmitted data keeping the quality of stream data and communication delays by compressing stream data. Some points are similar with our proposed method is reducing the communication delays and small data transmissions. One of the drawbacks of the method is that the data sources need to compress the data before their transmissions and this causes further delays.

In addition, some stream data analysis systems have been developed in ([2], [11]). However, the quality of stream data is fixed under these schemes. Our proposed approach in this paper improves the quality progressively.

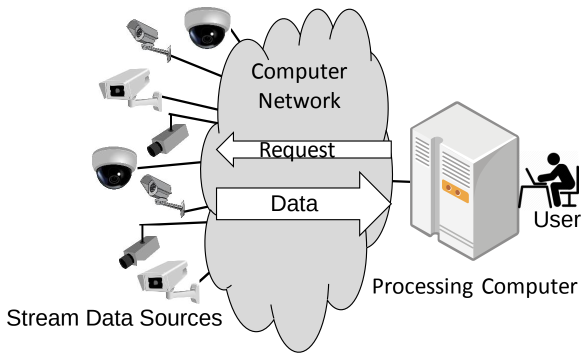

Fig. 1: Our assumed system architecture

3. Assumed System

In this section, we explain our assumed system.

3.1 System Architecture

Figure 1 shows our assumed system architecture. An user designates processes for continuously generated data (stream data) to a processing computer. The processing computer executes designated processes every data reception. Such a type of processes is called stream processing. The processing computer gathers necessary data for processes and executes processes continuously. The processing computer has a buffer for storing received data and execute processes for the data.

Some IoT devices such as surveillance cameras continuously get data about their observations such as video data and act as stream data sources. They and the processing computer connect to a computer network. In the cases that the network bandwidth is stable, the Internet can be assumed to be the computer network. The data sources and the processing computer can communicate with each other via the computer network. In this assumption, the data sources divide their generated stream data into smaller (we called those divided data are lower data qualities) and then send to the processing computer continuously. When the processing computer requests data (this case we called higgher data quality) to stream data sources, the requested data sources generate the requested data to their own buffer temporarily and then return to the processing computer sequentially. The processing computer receives the requested data to its own buffer and performs processing. By processing the requested data, we called progressive quality improvement.

3.2 Application Scenario

In this subsection, we introduce an application scenario.

Suppose an area in that some surveillance cameras are deployed and a processing computer gathers their recorded video data. They connect to a designated computer network and communicate with each other similar to our assumed system architecture.

As an example application scenario, we assume a person re-identification system by a face recognition. For this, the application designates the process that notifies to the user when the processing computer identifies person' face and determine whether the person is registered or not in the video data got from surveillance cameras. To detect faces, the user submits the face images of registered persons to the processing computer beforehand. The processing computer continuously analyzes image data got from surveillance cameras and identifies faces in received image data. When the processing computer finds faces in an image data, it checks whether the found faces are those of registered person or not. If the processing computer detects the faces that are not registered, it sends a notification to the user by e-mail or other messaging services.

3.3 Research Objective

In the scenario introduced in Subsection 3.2, the application performance is the probability to identify the person by face recognition. This can increase by frequently analyzing image data, since they are moving and the probability to record them in the video increases. Moreover, the probability to record person increase by collecting video data from more surveillance cameras.

However, communication capacities between stream data sources and the processing server are limited. Therefore, a more frequent data analysis and also a larger number of data sources cause a longer communication time. Longer communication time lengthens the delay from the time of data generation to the time to finish analyzing the data.

Therefore, our research objective is reducing the delay keeping the application performance.

3.4 Mathematical Definition

In this subsection, we explain a mathematical model for our assumed system. Suppose that the system has N stream data sources. These stream data sources cyclically send their observed data every C_n $(n = 1, \cdots, N)$ unit times. Let $D_{n,a}(t)$ denote the whole stream data at tth cycle. Whole stream data mean original stream data without dividing them into some qualities. The system can divide $D_{n,a}(t)$ into Q data $D_{n,q}(t)$ which is q th quality data of $D_{n,a}(t)$. The data amount of $D_{n,q}(t)$ is denoted by $S_{n,q}(t)$. $GT_{n,q}(t)$ denotes the genration time of $D_{n,q}(t)$ and $P_{n,q}(t)$ denotes the time required to process it. $ST_{n,q}(t)$ is the time to start processing $D_{n,q}(t)$ and $FT_{n,q}(t)$ is the time to finish processing it. Here, $FT_{n,q}(t) = ST_{n,q}(t) + P_{n,q}(t)$. Processing delays $Delay_{n,q}(t)$ are from $GT_{n,q}(t)$ to $FT_{n,q}(t)$ and given by the following equation:

$$Delay_{n,q}(t) = FT_{n,q}(t) - GT_{n,q}(t) \qquad (1)$$

In cases that the processing computer completes analyses, total processing delays for finishing analyses $Delay_n(t)$ is given by the following equation:

$$Delay_n(t) = FT_{n,Q}(t) - GT_{n,1}(t) \qquad (2)$$

The average processing delay for the data source n is:

$$\frac{1}{T}\sum_{t=1}^{T} Delay_n(t) \qquad (3)$$

Where T is the final cycle. The objective is maximizing the number of data sources to process, which corresponds to minimizing Equation 3.

We give the probability for processes to proceed to the next level $PProb_{n,p}(t)$ $(p = 1, \cdots, Q-1)$. For example, the probability to request $D_{1,2}(1)$ when the processing server finishes processing $D_{1,1}(1)$ is $PProb_{1,1}(1)$.

4. Proposed Method

In this section, we explain our proposed method.

4.1 Basic Idea

Generally, data have some qualities. Data analyses can be applied for each quality and data with the highest quality often give the best performance for analyses. For example, one of qualities for image data is resolution. Image data with 640x480 pixel sizes have a higher quality than image data with 320x240 pixel sizes. Image analyses to find faces can be applied for various pixel sizes while a higher resolution image data generally gives a higher accuracy. By finally analyzing data with the highest quality, applications can achieve the same performance. By using the progressive JPEG encoding, the image can be decoded at low quality when the smaller bytes are available. The image can be decoded at high quality when more bytes are available in addition to the bytes for the low quality image. When processing computers analyze data sequentially in the order of quality from the lowest to the highest, they can stop data analyses when the subsequent analyses for higher quality data are meaningless. For example, same as the example in the introduction section, suppose the case that a processing computer analyzes video data to detect faces. The processing computer first receives the lowest quality image data of a frame and analyzes the difference from the previous frame. In case that the difference are small, the processing computer skips the analyses of higher quality image data since new humans do not appear in the frame because of small difference. In such cases, the processing computer do not need to receive higher quality data when subsequent analyses are meaningless. Therefore, by analyzing data in the order of data quality and stop analyses when subsequent analyses are meaningless, processing computer can skip the receptions of higher quality data.

In cases that the probability to proceed to higher quality data analyses is small, the total amount of received data is reduced, compared with the case that all quality data are received. Generally, the amount of a lower quality data is smaller. Therefore, the data amount to be received is reduced when the probability is small compared with the processing computer receives the highest quality data without consideration of data qualities. Thus, communication delays are reduced keeping the application performance. We call this approach *progressive quality improvement* approach.

4.2 Data Stream Processing

In this subsection, we explain the processes of data collection in the conventional approach and our proposed approach. In this example, the data sources are two cameras.

We first explain data streams processing under the conventional approach. Camera 1 sends its recorded video data to the processing computer. It sends each image data for a frame ev-

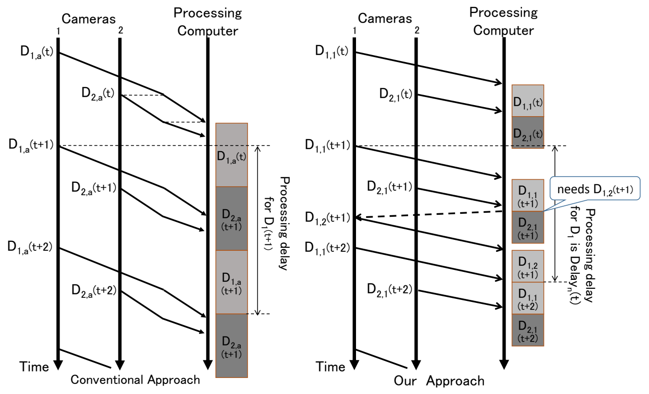

Fig. 2: Stream data collection of an conventional approach and of our proposed approach

ery getting it. In the Figure 2, the t th frame data is shown by $D_{1,a}(t)$ $(t = 1, \cdots, T)$. For example, when the frame rate is 10Hz, Camera 1 sends a frame data every 0.1 sec. Hence, $GT(D_{1,a}(t+1)) = GT(D_{1,a}(t) + 0.1)$. In addition, Camera 2 sends its recorded video data to the processing computer. In this example, the frame rate for Camera 2 is the same as that for Camera 1, but the time to start sending the video data differs. After Camera 1 sends $D_{1,a}(t)$, Camera 2 sends $D_{2,a}(t)$. While both Camera 1 and Camera 2 send data, the input communication bandwidth for the processing computer is divided into them. Therefore, the communication speed of Camera 1 decreases as shown in the Figure 2. After Camera 1 finishes sending $D_{1,a}(t)$, the input communication bandwidth is dedicated for the communication with Camera 2 and the communication speed of Camera 2 increases as shown in the Figure 2. When the processing computer finishes receiving $D_{1,a}(t)$, it starts processing $D_{1,a}(t)$. While processing $D_{1,a}(t)$, the processing computer finishes receiving $D_{2,a}(t)$. Since the processing computer processes $D_{1,a}(t)$ at this time, it remains the received $D_{2,a}(t)$ to its communication buffer and starts processing it after finishing processing $D_{1,a}(t)$. Similarly, while processing $D_{2,a}(t)$, the processing computer finishes receiving $D_{1,a}(t+1)$. The processing computer starts processing it after finishing processing $D_{2,a}(t)$. The processing delay for $D_{1,a}(t+1)$ in this case is shown in the Figure 2. This is the time from the start of sending $D_{1,a}(t+1)$ to the finish of processing $D_{1,a}(t+1)$.

Next, we explain data streams processing under our proposed approach. Similar to the example for the conventional method, Cameras 1 and 2 send their recorded image data to the processing computer cyclically. Different from the conventional method, the image data is divided into 2 levels. For example, the data for the first level is the lowest quality image data and the data for the second level improves the quality of the data. We assume that the data for the second level only includes the difference data from the first level and that the amount of the data for each level is the same. To make the example simple, we assume that the data amount of each level are just half of the data amount of $D_{1,a}(t)$ $(t = 1, \cdots, T)$. Therefore, the time needed to send $D_{n,q}(t)$ $(n = 1, 2, q = 1, 2)$ is the half of the time needed to send $D_{n,a}(t)$. Therefore, the communication of $D_{1,1}(t)$ do not overlap that of $D_{2,1}(t)$ though the communication of $D_{1,a}(t)$ overlaps that of $D_{2,a}(t)$. The processing computer does not request the second level data in the first cycle. An example of the reason not to request the second level data is that the difference of the image data from the previous image data is not so large. In the $t+1$ th cycle, the processing computer starts processing $D_{1,1}(t+1)$ after finishing receiving it. After processing $D_{1,1}(t+1)$, the processing computer requests the second level data. An example of the reason to request the second level data is that the difference of the image data from the previous image data is large. When Camera 1 receives the request for $D_{1,2}(t+1)$, it starts sending $D_{1,2}(t+1)$. To make the example simple, the processing computer does not request the second level data of $D_{2,1}(t+1)$ in this case. After receiving $D_{1,2}(t+1)$, the processing computer processes it and finish the data analysis of $D_1(t+1)$. The processing delay for $D_1(t+1)$ in this case is shown in the Figure 2. This is the time from the start of sending $D_{1,1}(t+1)$ to the finish of processing $D_{1,2}(t+1)$. The $D_{1,2}(t+1)$ includes only the data different from $D_{1,1}(t+1)$. By combining $D_{1,1}(t+1)$ and $D_{1,2}(t+1)$, we can get the higher data quality.

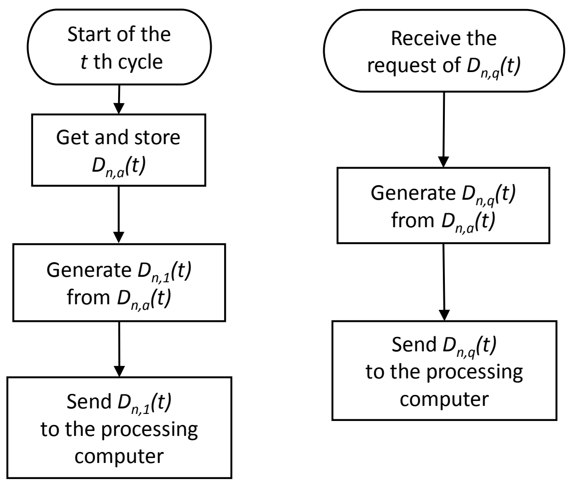
Start of the t th cycle
Get and store $D_{n,a}(t)$
Generate $D_{n,1}(t)$ from $D_{n,a}(t)$
Send $D_{n,1}(t)$ to the processing computer
Receive the request of $D_{n,q}(t)$
Generate $D_{n,q}(t)$ from $D_{n,a}(t)$
Send $D_{n,q}(t)$ to the processing computer

Fig. 3: Data sources generation

In this case, the processing delay under our approach is shorter than that under the conventional approach since the processing time for $D_2(t)$ is reduced.

4.3 Our Proposed Method

In this subsection, we explain the algorithms for our proposed approach.

4.3.1 Algorithm for Data Sources

Figure 3 shows the flow chart of data sources. When the t th cycle starts, each data source n gets $D_{n,a}(t)$ from their sensors and stores it to their storages temporary. First, they generate $D_{n,1}(t)$ from $D_{n,a}(t)$ and send $D_{n,1}(t)$ to the processing computer.

When the data source n receives the request of $D_{n,q}(t)$, it generates $D_{n,q}(t)$ from stored $D_{n,a}(t)$ and sends $D_{n,q}(t)$ to the processing computer.

4.3.2 Algorithm for Processing Computer

Figure 4 shows the flow chart of the processing computer. When the processing computer receives $D_{n,q}(t)$, it processes $D_{n,q}(t)$. When $q = Q$ and $D_{n,q}(t)$ is the final quality data,

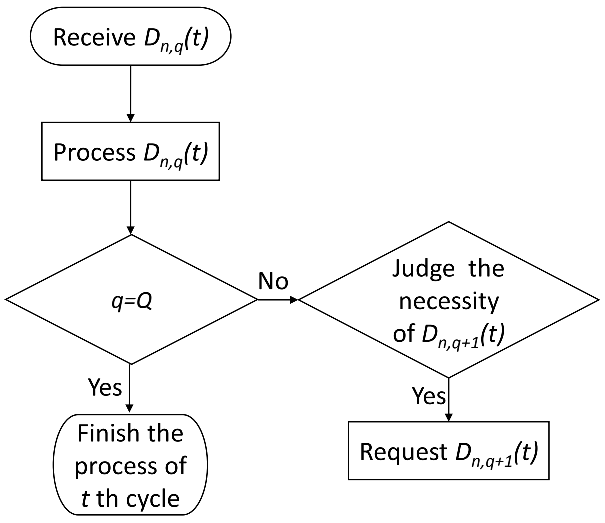

Fig. 4: Data processing in the processing computer

the process of t th cycle finishes. Otherwise, the processing computer judges the necessity of $D_{n,q+1}(t)$. In case that $D_{n,q+1}(t)$ is needed for the process execution, the processing computer requets $D_{n,q}(t)$ to the data source n.

4.3.3 How to Divide Data

In our proposed method, each stream data is divided into some qualities. We suppose two approaches to divide data.

The first one is the cases that a higher quality data can be constructed by combining some data. For example, image data with 640x480 pixel sizes can be constructed by 4 image data with 320x240 pixel sizes. In this case, the data amount of the q th quality data of the stream data n at the t th cycle is given by:

$$\sum_{i=1}^{q} S_{n,i}(t) + \alpha_{n,i}(t) \tag{4}$$

Here, $\alpha_{n,i}(t)$ is an overhead caused by combining data.

The second one is the cases that a higher quality data is constructed separately. For example, it is difficult to fully decode image data with 640x480 pixel sizes by combining compressed 4 image data with 320x240 pixel sizes. In this case, the data amount of the q th quality data of the stream data n at the t th cycle is given by $S_{n,q}(t)$.

The application of the stream processing system selects appropriate method to divide data.

5. Evaluation

In this section, we show evaluation results of our proposed method by using our developed simulator.

5.1 Evaluation Setup

In this evaluation, we assume the application explained in Subsection 3.2 and use the parameters shown in Table 1. Input Bandwidth is the input communication bandwidth for the processing computer. When the processing computer communicates with some data sources, the input bandwidth is fairly shared among data sources. Output Bandwidth is the output communication bandwidth of each data source. Total data amount is the data amount of $D_{n,a}(t)$ ($n = 1, \cdots, N$, $t = 1, \cdots, T$). To make the evaluation results easily understandable, we set the same data amount for all data items. We use the first method explained in the section 4.3.3 to divide data into some qualities and set $\alpha_{n,q}(t) = 0$. Processing Time Ratio is the ratio to define the processing time. The value is $P_{n,a}/S_{n,a}$. We set these parameters considering practical situations.

Table 1: Parameter values

Input Bandwidth	10 [Mbps]
Output Bandwidth	10 [Mbps]
Total Data Amount	12.5 [Kbytes]
Processing Time Ratio	10^{-6}

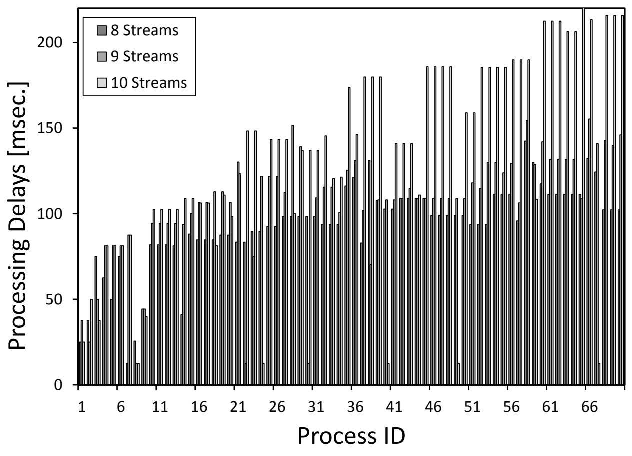

Fig. 5: Processing delays of each process

We use the same values for $PProb_{n,p}(t)$ ($p = 1, \cdots, Q-1$). For this, we set the final probability $FProb$ for processes to proceed to the final level. $PProb_{n,p}(t) = FProb^{1/N}$.

We simulate the stream processing system for 60 seconds and get the processing delays.

5.2 Processing Delays

The objective of our proposed method is reducing average processing delays. Before taking average values of processing delays, we first check processing delays for each process.

Figure 5 shows the evaluation result. The horizontal axis is process IDs. Process IDs are given to the processes that continues to the final level. Process IDs are given separately to each data source. Thus, all processing delays shown in the figure are values of $Delay_n(t)$. The vertical axis is processing delays for each process. We show the processing delays for the cases that the number of the streams is 8, 9, and 10. In this evaluation, the intervals of data streams are the same and are 100 [msec.] We set the final probability to 0.7 and the number of the levels is 2 as just an example.

The processing delays for the case of 8 streams increases in small process Ids (approx. until 10th ID). After that the processing delays saturate and are almost constant. The system can continue to process stream data sent from data sources. The processing delays for the case of 9 streams tend to a little longer than that of 8 streams, but saturate similar to the case of 8 streams. However, in the case of 10 streams, the processing delays tend

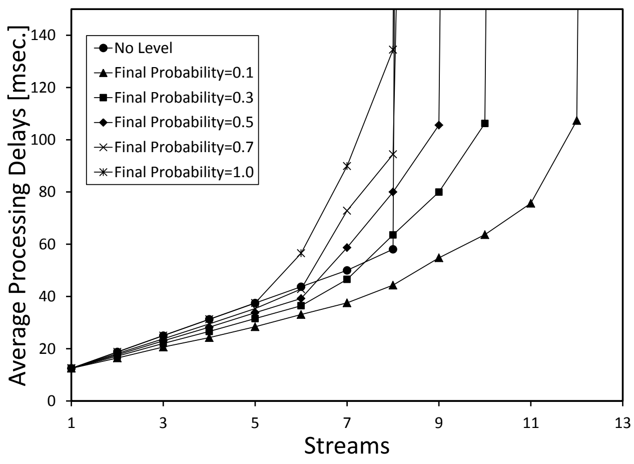

Fig. 6: Average processing delays under different final probability

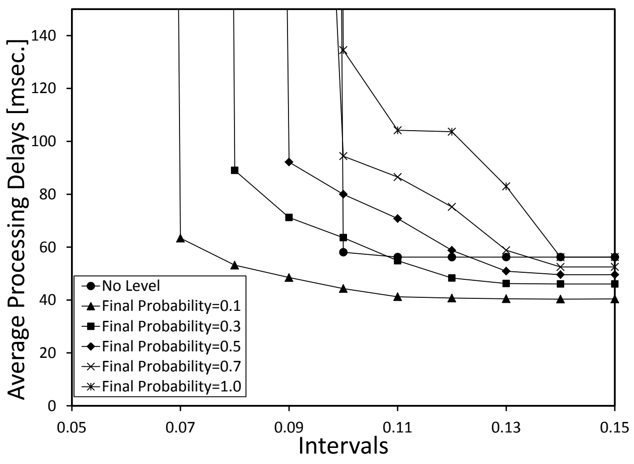

Fig. 7: Average processing delays of different intervals

to increase along with the process IDs. This is the probability that the processing computer finish processing before receiving the next stream data to process is small and the stream data stored in the buffer of the processing computer increases. Since the processing delays continue to increase, the system will fail in this case. Therefore, the user of the system should set the number of the streams less than 10 under keeping other parameters.

5.3 Influence of Number of Streams

The data amount that the processing computer receives increases as the number of streams increases. Thus, the average processing time is influenced by the number of streams. We investigate the influence.

Figure 6 shows the result of the average processing delays changing the number of streams. The herizontal axis is the number of streams and the vertical axis is the average processing delays. We simulate the processing delays under different final probabilities, the final probability is 0.1, 0.3, 0.5, 0.7, 1.0. No Level (represents the case of conventional approach). The intervals for all data streams are the same and is 0.1 [msec.] The number of qualities under our proposed method is 2.

A larger final probability values causes a longer delay since the processing computer finishes processing before receiving the next stream data to process is large and the stream data stored in the buffer of the processing computer increases. The average processing delays under the conventional method increases proportionally as the number of streams increases for the cases where the number of streams is less than 8, because the data amount that the processing computer receives increases proportionally. For the cases where the number of streams is larger than 8, the average processing delays increase sharply. This is the processing delays increase as the time proceeds as shown in the case of 10 streams in Figure 5 and the system fails to process stream data continuously. We can see similar phenomena for the cases of our proposed method. However, the maximum number of streams that the system works is larger compared with the conventional method. For example, in the case where the final probability is 0.5, the average processing delay sharply increases when the number of streams is 10. Therefore, the processing computer can collect data from more data sources by using our proposed method compared with the conventional method.

The average processing delays in the cases of the maximum number of streams just before the system fails under our proposed method are longer than that under the conventional method because the processing computer receives a higher quality data after requesting it to data sources in our proposed method. For example, in the case where the final probability is 0.5, the average processing delay is 106 [msec.] of 9 streams under our proposed method though it is 58 [msec.] of 8 streams under the conventional method. This is a demerit of our proposed method.

5.4 Influence of Intervals

The processing computer frequently receives data as the interval shortens and the average processing time increases. Therefore, we investigate the influence of the intervals.

The result of the evaluation is shown in Figure 7. The herizontal axis is the interval values and the average of processing delays is in the vertical axis. In this evaluation, the number of streams is 8. For our proposed method, the number of levels is 2.

Similar to the previous evaluation result, the system fails when the interval is excessively short and the average processing time sharply increases. In the conventional method, the average processing delays are almost constant when the interval is larger than 0.1 [msec.]. This is the processing delays sharply increase in the cases where the processing computer receives the next data during processing in the conventional method. In of our proposed method, the shortest interval that the system works is shorter compared with the conventional method. For example, in the cases where the final probability is 0.5, the average processing delay sharply increases when the interval is 0.08. Therefore, the processing computer can collect data with a shorter interval by using our proposed method compared with the conventional method.

5.5 Influence of Number of Levels

The number of levels influences the average processing time. We investigate the influence.

In the Figure 8, shows the average processing delays under different number of the levels. The horizontal axis is the number of levels and the vertical axis is average processing delays. The

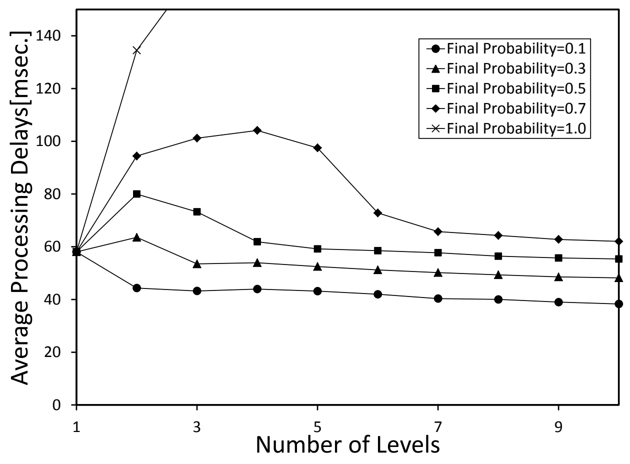

Fig. 8: Average processing delays of different stream levels

intervals value is 0.1 [msec.] and the number of streams is 8.

The average processing delay for the case that the number of levels is 1 represents the average processing delay under the conventional method. The results otherwise are that under our proposed method. When the number of levels is small, the average processing delays increase as the number of levels increases except for the case that the final probability is 0.1. When the number of level is large, the average processing delays decease as the number of levels increases. This is the processing computer should wait for the reception of the requested data and the processing delay increases when the number of levels is small. However, when the number of levels is large, the processing computer can avoid redundant reception of data and the processing delay decreases. The effectiveness of the reduction of the processing delay is large when the number of levels is large. Accordingly, the average processing delays increase under small number of levels and decrease under large number of levels.

6. Conclusion

The analysis frequency is one of the main factors to improve performances of some IoT applications. To improve the analysis frequency, we propose an efficient data collection scheme. In our proposed scheme, only the cases where higher quality data are needed for analyses, processing computers progressively collect them. Our simulation evaluation revealed that our proposed scheme can reduce the communication delay while keeping the application performances.

In the future, we plan to propose a scheme for the situation there are some processing computers. In additon, we will consider parallel processing of collected data.

Acknowledgement

This work was supported in part by JSPS KAKENHI Grant Numbers JP15H02702, JP17K00146, and JP18K11316. Also this work include the result of NICT Osaka University joint research ”research and development of advanced network platform technology for large scale distributed computing”.

References

[1] A. Akbar, G. Kousiouris, H. Pervaiz, J. Sancho, P. Ta-Shma, F. Carrez, and K. Moessner, Real-time probabilistic data fusion for large-scale iot applications, *IEEE Access*, vol. 6, pp. 10015-10027, 2018.

[2] C. H. Lu, C. H. Yu, B. H. Chen, I. S. Hwang, and S. S. Huang, Semi-supervised data stream analytics with balanced recognition performance and processing speed, in *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pp. 355-356, June 2017.

[3] J. A. Colmenares, R. Dorrigiv, and D. G. Waddington, A single-node datastore for high-velocity multidimensional sensor data, in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 445-452, Dec 2017.

[4] J. C. Beard and R. D. Chamberlain, Analysis of a simple approach to modeling performance for streaming data applications, in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 345-349, Aug 2013.

[5] J. Xu, Y. Andrepoulos, Y. Xiao, and M. van der Schaar, Nonstationary resource allocation policies for delay-constrained video streaming: Application to video over internet-of-things-enabled networks, *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 782-794, April 2014.

[6] P. Zhao, W. Yu, X. Yang, D. Meng, and L. Wang, Buffer data-driven adaptation of mobile video streaming over heterogeneous wireless networks, *IEEE Internet of Things Journal*, pp. 1-1, 2017.

[7] S. Molina-Giraldo, H. D. Insuasti-Ceballos, C. E. Arroyave, J. F. Montoya, J. S. Lopez-Villa, A. Alvarez-Meza, and G. Castellanos-Dominguez, People detection in video streams using background subtraction and spatial-based scene modeling, in *2015 20th Symposium on Signal Processing, Images and Computer Vision (STSIVA)*, pp. 1-6, Sept 2015.

[8] S. R. V and M. Dakshayini, Priority based optimal resource reservation mechanism in constrained networks for iot applications, in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 1228-1233, March 2016.

[9] T. Buddhika and S. Pallickara, Neptune: Real time stream processing for internet of things and sensing environments, in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1143-1152, May 2016.

[10] U. A. Agrawal and P. V. Jani, Performance analysis of real time object tracking system based on compressive sensing, in *2017 4th International Conference on Signal Processing, Computing and Control (ISPCC)*, pp. 187-193, Sept 2017.

[11] Y. Ge, X. Liang, Y. C. Zhou, Z. Pan, G. T. Zhao, and Y. L. Zheng, Adaptive analytic service for real-time internet of things applications, in *2016 IEEE International Conference on Web Services (ICWS)*, pp. 484-491, June 2016.