Automatic Deadline-Oriented Sampling Method for Coarse-grained Stream Processing

Sunyanan Choochotkaew, Hirozumi Yamaguchi, Teruo Higashino
Graduation School of Information Science and Technology
Osaka University, Osaka, Japan
Email: {sunya-ch,h-yamagu,higashino}@ist.osaka-u.ac.jp

***Abstract*—Towards IoT-enabled world, the response time starting from data occurrence at the source until processed data delivery to the actuator is another QoS metric to be concerned. We call this requirement deadline. In coarse-grained stream processing, we could partially drop data in the streams with a specific drop rate to meet the deadline. This paper proposes an autonomic sampling method to decide the drop rate aiming at response time reduction oriented by the user-specified deadline. With consideration of processing and communicating time sharing among distributed worker nodes, we calculate a sampling number to satisfy the deadline requirement while preserving the maximum drop rate. The device will set a goal to maintain this sampling number for the next operating window. To evaluate the performance, we have implemented the proposed method on top of our previously-proposed stream processing engine called EdgeCEP. The results present that our proposed method can reduce almost 2-times latency and preserve a higher amount of request outputs compared to the fixed rate approach.**

I. Introduction

Driving a new era of life-quality services, stream processing technology plays an essential role to extract useful knowledge from overwhelming and never-ending streaming data generated by IoT devices. With stream processing, you can ask the system to monitor your house entrance for 24 hours and automatically notify you at the moment when it found strangers come around. From learning your daily-life habits, it may be able to assist you to change your behavior and improve your health. Individually, each processing application has a different specified requirement of prospective result delivering known as *Quality of Service* (QoS) [1]. Real-time applications require processing within a short and stringent deadline. Some of those are not concerned only about time-based metrics like throughput and latency but also care the content-based values such as result quality (accuracy), for example, aircraft control systems. We consider this kind of processing as a fine-grained type. To archive that, the straightforward way is increasing the computational power of processors. Additionally, smart scheduling and distributing processing tasks over multiple computing modules is also a potential solution. On the other hand, a coarse-grained stream processing has more flexibility to adjust the sampling rate by ignoring some data to preserve its deadline (latency), for example, video surveillance systems.

Since the 1990s, Quality of Service (QoS) has been observed and studied starting from the field of telephony and the data networking area. In the data-collecting system, adaptive sampling method and load-shredding specification are considered as potential solution techniques to maximize the entire QoS property of networks [2] [3] [4]. In the same way, both techniques are integrated into the stream processing framework additionally to the scheduling module to improve or guarantee user-specified QoS requirements. They are also known as approximation techniques [5]. The basic idea is to set the sampling rate that can minimize an error from shedding real-time constraint [6]. Still, to guarantee on time-based QoS, especially latency metric, for the whole path of distributed stream processing faces an additional challenge due to a communication gap between processors. Only a real-time constraint at each node cannot guarantee the total latency of processing and delivering at the final consumers.

In this paper, we point out the quality-improvement potentiality of communication factors in the load shedding problem for coarse-grained stream processing across distributed collaborative processors. We introduce an application-level respond-time metric to determine the quality of service in stream processing application. To handle fluctuation of data flow as well as a communication link, we propose an automatic deadline-oriented sampling method implemented on our previously-proposed *EdgeCEP*, a fully-distributed complex event processing system [7] [8]. Responding to the feedback from neighbor nodes, it computes the number of sampling in a long-ranged window. During an operating time, the sampling rate is adjusted to preserve the target sampling number. Additionally, we modify the learning-based prediction for processing time, previously-proposed in [9], to determine the proper sampling number. According to the experimental results, without shedding, and fixed-rate shedding incurs about 2s delay over 1s requirement with 0.5 accuracy threshold. Meanwhile, the proposed method lets only 0.65s delay occur.

The rest of the paper is organized as follow. Section II gives background knowledge and discusses related works. Section IV introduces the EdgeCEP system and Section IV describes the proposed method. Section V presents experimental settings and results. Section VI states some remaining issues, and finally, Section VII concludes the paper.

II. Related Work

A technology to process continuous and timely flowing information, such as video stream, is well-known as information flow processing or stream processing. Since 1988, numerous general-purpose engines have launched in various manners [10]. In common, we can categorize them from their processing methods: either tuple-based or object-based. The former is similar to the continuous database query and processing [11] [12] while the latter is preferable for detection [13]. Formerly, they are called Data Stream Management System (DSMS)

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

and Complex Event Processing (CEP), respectively. However, DSMS-originated tuple-based method is sometimes also called CEP due to an enhancement of detection efficiency [14]. Recently, in our previous work, we propose a combination approach, named *EdgeCEP*, to perform tuple-based CEP at edge devices in a fully distributed manner [7] [8].

Since a decade ago, the Quality of Service (QoS) concept comes up as an alternative feature in stream processing applications. In [5], the authors classify QoS metrics into two broad aspects: time-based and content-based. The former includes throughput and delay (latency) while the latter refers to drop rate, sliding window size, approximation quality, and mining quality. The definition of latency is limited to the time distance from input arrival till output processed. The QoS-oriented method could be either best effort or guarantee. The best-effort approach tries to maximize quality value. Meanwhile, the guarantee-approach further requires negotiation to obtain the exact requirement satisfaction. Most of existing stream processing engines support the best-effort QoS management with various kinds of approaches. One solution is to optimize a task-execution plan over a considerable number of processors running in parallel. The better plan usually reduces accomplishing time and resource consumption. In *EdgeCEP*, we assign the task with optimizing global flow volume. In *Tasklet*, system users are allowed to provide resource requirements, such as reliability and speed, for their task offloading [15]. Unlike fine-grained processing tasks, where all data are significant, coarse-grained tasks allow dropping some data in streams due to an abundant amount or unneeded content. For such a kind of tasks, there is further applicable solution called load shedding for reducing the inputting data.

For load shedding, there are multiple ways of solutions. Some prioritize the low-frequent streams over high-frequent ones. For instance, ref. [3] modify a signal strength of transmitter according to its flowing frequency. Also, we may prioritize the data tuples by directly referencing to user-specific requirements of information, such as thematic and spatiotemporal relevance [16]. Another technique is to sample the data streams randomly. Ref. [17] introduces applying basic sampling techniques such as *Bernoulli* and *Reservoir*. *Bernoulli* is faster and simpler with a fixed sampling rate but cannot control the variability of sample size while *Reservoir* controls the sampling size instead. In [6], the sampling size is set to achieve real-time constraints with minimum error. Additionally, the paper introduces the algorithm to decide on shed-operator positions overlaying an execution plan. Similarly in [18], the *Drop* operators have been implemented in *Aurora*, one of well-known stream processing engines. The drop operations can be either random drop with a fixed rate or semantic drop with utility determining. In [19], the sampling rate is adapted exploiting fuzzy logic and regression techniques.

The contributions of this paper are summarized as follows:

1) We introduce a response-time metric of QoS in stream processing to determine the time difference starting from source, processing at processors, and reaching the destination. We consider the deadline as a requirement of this metric.
2) Considering both processing and communicating latency, we approximate a target sampling number aiming at deadline requirement, at the same time, preserving the maximum drop rate requirement. Furthermore, we exploit the learning-based model to determine processing latency according to a variety of processors, functions and inputting data.
3) We propose an autonomic adaptive sampling mechanism controlling the number of sampling instead of the sampling rate, and operating in a distributed way through the feedback message.
4) We implement and evaluate our proposed method in real devices on top of the *EdgeCEP* system. The results show superior benefits over baseline (no dropping) and fixed sampling rate method.

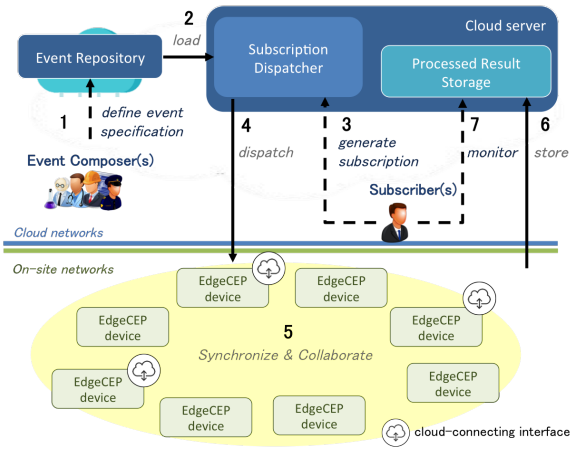


Fig. 1: EdgeCEP Framework

III. EdgeCEP: Fully-distributed Complex Event Processing on IoT Edges

EdgeCEP has been developed as a general complex event aiming at operating on smart IoT edge devices in self-organized distributed manner [7] [8]. The framework is shown in Fig. 1. Data to be processed in the systems are representing as flows of *Events*. The specification of an event is pre-defined and globally provided with definition, $Name(Att_1, ..., Att_n)$, by event composers (mostly considered as specialists). We consider the events with detecting, processing, and producing behaviors as *Composite* events, specified in the following language structure.

define	$Name(Att_1, ..., Att_n)$ [aggr] [every T]
$< pre >^*$	
[**case** n:]	
detect	$Pattern(content_1, ..., content_m)$
assign	$attr_1 = f_1, ..., attr_p = f_p;\ f = F(content)$
[**consuming**	$e_1, ..., e_h; e_i \in content$]
$< post >^*$	
where*	$Att_1 = g_1, ..., Att_n = g_n;\ g = G(attr)$
[***group by***	$(location \vert srcid)$]

*only for aggregation specification (aggr)

We identify the detecting behavior with **detect** key which consists of interest pattern of event contents. We identify the processing behavior with **assign** key which consists of processing function and input contents. For the producing behavior,

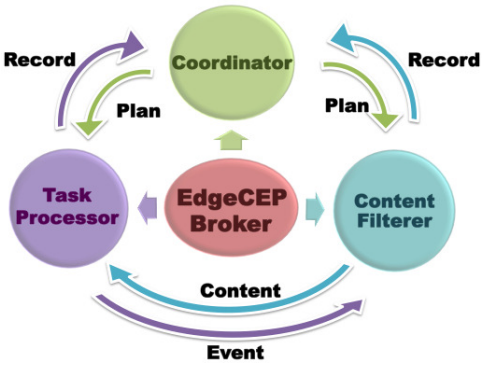


Fig. 2: EdgeCEP Components

we describe producing and consuming policy with **every** and *consuming* keys, respectively. The term "content" denotes conjunction of events with values that satisfy the specific conditions. For instance, the content, "Temperature.$val > 40°C$", refers a "Temperature" event with a condition "$val > 40°C$". Correspondingly, the temperature event with value more than $40°$, for example, $45°$, will be filtered in this content group. Event composers can specify the detecting and processing behaviors for the same subscription by multiple interest patterns using the keyword **case**. For aggregation, there is an aggregation keyword, that is **aggr**. If it is aggregation type, the composers must separate the specification into two parts: $< pre >$ and $< post >$ in the same way as the map and reduce method.

Users of the system, subscribers, provide a set of continuous queries, called subscriptions, via the cloud server. The specification structure of subscription is similar to the composite event except for the *define* key. Instead of attributes, we need to declare the requested title and actuator destination, as follow:

subscribe $Title$ **from** $UserID$ **activate** $Actuator$ [aggr] [every T]

User-queried subscriptions can only refer to available event definitions in the event repository, uploaded by event composers. The cloud server will dispatch the valid subscriptions to the network of collaborating EdgeCEP devices, called brokers. Brokers could be wired or wirelessly connected to sensors and actuators. Sensors sense surrounding environments while actuators activate a corresponding action. In the same time, they also act as distributed processing nodes. Note that, the running results may be uploaded to the cloud for monitoring by subscribers.

As an EdgeCEP broker, there are three functional modules parallel operating inside the device, as presented in Fig. 2. The content filterer module exploits the publish-subscribe mechanism and hashing method to feed an event to the corresponding agents in the task processor module, i.e., performing detecting behavior. The task processor modules consist of multiple processing agents. Each of them is responsible for generating a composite event or producing an output of subscriptions. The processing and producing operations perform here. The last module, coordinator, is for monitoring runtime status, sharing

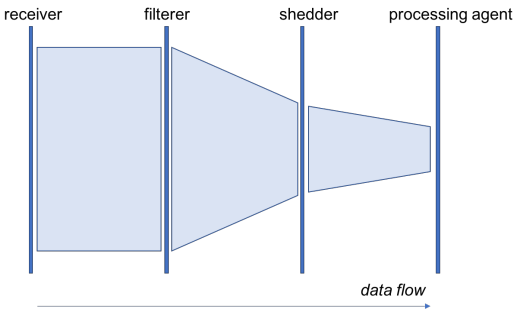


Fig. 3: Flow Shedding Concept

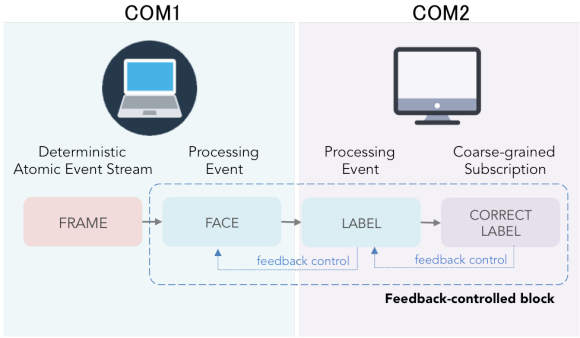


Fig. 4: Feedback Mechanism

that knowledge with its neighbors, and deciding on task processing and forwarding plan to complete user subscriptions.

IV. An Automatic Deadline-Oriented Sampling

For the filter-enabled systems like EdgeCEP, the sampling should run after the filter operation due to the data flow after filtering are supposed to be semantically equal. The flow shedding concept is illustrated in Fig. 3. The proposed method performs sampling decision in a distributed manner using feedback control block from the further brokers. To illustrate, in Fig. 4, the correct label subscription set actuator at **COM2** and the video frame stream is generated at **COM1**. As a broker, **COM1** decides to assign the *FACE* composite event to itself and forward the rest processing to **COM2**. Then, it automatically determines sampling number of *FRAME* based on current latency which is estimated by self-observation and future latency deriving from **COM2**'s *LABEL* feedback.

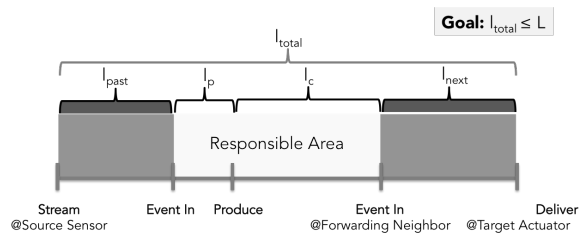


Fig. 5: Responsible Area for Latency Control at Individuals

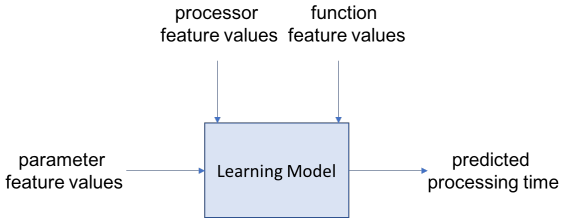


Fig. 6: Processing-Time Prediction Concept

Input: L,N, N_{max}, l_p, l_c, l_{past}, l_{next}, **ML()**, R_{th}
Output: N
$N_{candidate} \leftarrow$ **ROT**(N, l_p, l_c)
$N_{min} \leftarrow (1 - R_{th}) \times N_{max}$
if ***ML***$(N, l_{past}, l_{next}, L)$*=satisfied* **then**
  $N \leftarrow N_{candidate}$
else
  $N \leftarrow N_{min}$
end

Algorithm 1: Sampling Decision Procedure

A. Problem Definition

Suppose user specify deadline L for the response time with maximum drop rate threshold R_{th}, the current total latency, l_{total}, can be estimated by summation of the following values: (1) past latency, l_{past}, counting from source to arrival at the considering node, (2) future latency, l_{next}, provided from the next neighbor, (3) processing latency, l_p, and (4) communicating latency, l_c (see Fig. 5). Provide function $F(N)$ and $G(N)$ to determine l_p and l_c, respectively, from the sampling number, N, out of total number, N_{max}.

Find N that

$$l_{total}(N) \leq L \text{ subject to } \frac{N}{N_{max}} \geq 1 - R_{th}$$

where $l_{total}(N) = l_{past} + l_{next} + F(N) + G(N)$

B. Sampling Number Decision

The sampling number decision making will be performed periodically with a specific time window following the procedure in Algorithm 1. To decide the sampling number in each window, we exploit the learning-based approach, proposed in [9] for processing time functions. With the features processor, function, and parameter, we can estimate whether the processing time will satisfy the deadline constraints or not from the sufficiently-learning model (Fig.6). The features of processors and functions are mostly static except available memory and parameter size. To reduce the computation time, we determine the candidate number N from a simple computing function using the rule of three (ROT). Then, we test the candidate number, $N_{candidate}$, with the learning model (ML). If it is not satisfied, the minimum N will be computed from R_{th} and potential maximum input N_{max}.

C. Sampling Decision Procedure

Corresponding to the sampling number, N, the expected sampling probability, P_{exp}, is N/N_{total}. A broker will decide whether to sample each filtering-in event or not with this probability. We present the decision procedure in Algorithm 2. At each decision point, the recorded left quota, N_{left} of N, and the current filtering-in number, N_{cur}, are available. If the sampling probability is infinity owning to a long-enough deadline, it will always sample any incoming events. On the other hand, if there is no left quota, it will reject all coming-after events. For the rest cases, if the current sampling probability, i.e., $(N - N_{left})/N_{cur}$, is lower than P_{exp}, the considering event will be selected. Otherwise, it will be randomly picked up with P_{exp} probability.

Input: N, P_{exp}, N_{left}, N_{cur}
Output: *isSampled*
if R_{exp}=∞ **then**
  isSampled $\leftarrow$ true
else if N_{left}*=0* **then**
  isSampled $\leftarrow$ false
else
  $N_{sampled} \leftarrow N - N_{left}$
  curR $\leftarrow N_{sampled}/N_{cur}$
  if *curR* $< R_{exp}$ **then**
    isSampled $\leftarrow$ true
  end
  ramdom $\leftarrow$ Random $[0, 1]$
  if *random* $\leq R_{exp}$ **then**
    isSampled $\leftarrow$ true
  else
    isSampled $\leftarrow$ false
  end
end

Algorithm 2: Sampling Decision

V. Evaluation

We implement the proposed automatic deadline-oriented sampling method on top of currently-developing EdgeCEP in Java platform. The subscription is a coarse-grained type with an additional attribute of quality requirement. We compare our method with baseline without any sampling method (none), and with fixed-rate sampling method (fixed). We set the environments for evaluation with the scenario shown in Fig. 4. The details are as follows.

A. Evaluation Setting

The subscription of correct label is represented in EdgeCEP specification language as:

subscribe	*Correct Label* **from** *User1* **activate** *COM2*
detect	*Label.distance* $<$ *2000*
assign	*CorrectLabel=Label*
consuming	*Label*

The function and processor features are listed in Table I, referring to [9]. We set the deadline requirement at 1s and create the connection between two processors with an ad-hoc link. We apply MultilayerPerceptron (mlp) classifier function to determine the satisfaction of processing time since it yields the smallest root relative squared error according to the current dataset. The *paramSize* and *availableMem* features will be fulfilled at runtime. The recalculating window time for sampling number decision is the 20s. The source stream is

TABLE I: Processing-time Learning Features

Function Features	**Face Detection**	**Face Recognition**	**Basic Function**
callCount	32	23	1
codelength	1,544	1,470	100
includedClassLen	28,420	91,844	8,570
varSize	608	608	32
Processor Features	**COM1**	**COM2**	
benchmarkTime	14,000 μs	11,731 μs	
clockSpeed	2,600 MHz	2,900 MHz	
CPUCore	8	4	
Requirement Feature	**Correct Label**		
deadline	1000 ms		
Input Features	**Video Stream**		
paramSize	varied in runtime		
availableMem	varied in runtime		

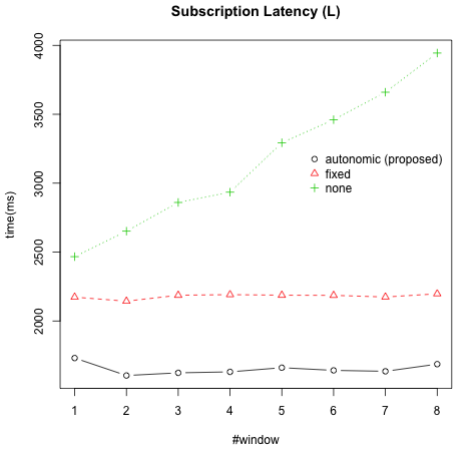


Fig. 7: Total Latency/Output for 8-windows run

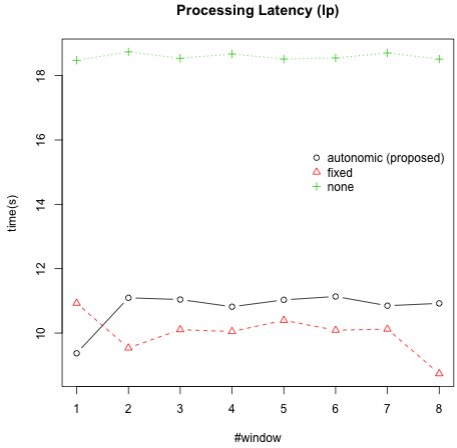
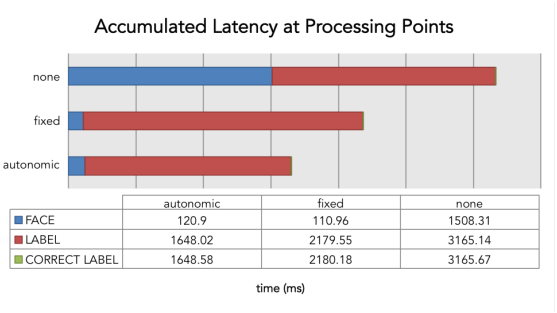


Fig. 8: Accumulated Processing Latency for 8-windows run



	autonomic	fixed	none
FACE	120.9	110.96	1508.31
LABEL	1648.02	2179.55	3165.14
CORRECT LABEL	1648.58	2180.18	3165.67

Fig. 9: Accumulated latency at each processing point

almost-one-minute recorded video. The maximum drop rate is 0.5. The sampling probability of the fixed-rate method equals to maximum drop rate.

B. Results

Fig. 7 presents the total response time (subscription latency) result. The latency on each window piles up over time since the without-sampling method (none) causes latency more than real-time constraints. Meanwhile, with-sampling methods (automatic and fixed) preserve a stable response time. The proposed method (autonomic) gains the closest response time to the user-specified requirement. It is over the deadline only 0.65s in average while *none* and *fixed* approaches are over the deadline 1.18s and 2.16s, respectively. The accumulated processing latency for each 20s window of each approach is shown in Fig. 8. *None* approach takes almost 20s for processing latency. Accordingly, this explicitly shows that processors could not deal with the subscription in realtime with the baseline approach.

time (μs)	l_c	l_c/l_p
Autonomic Sampling Method	1,950	13.82
Baseline	2,404	24.92

TABLE II: Communicating time over Processing time(l_c/l_p)

Table II shows the relation between communicating latency and processing latency derived from the experiment. Autonomic sampling can reduce the communicating latency almost 20% from the baseline. Also, the communication latency affects more in the total latency for both cases ($l_c >> l_p$). We present the average of accumulated latency at each processing point in Fig. 9. The sampling approach can significantly reduce the processing time at the first processing point compared to the baseline. Furthermore, the autonomic approach is superior to the fixed-rate approach at the rest processing point.

Since the number of output is directly corresponding to the number of input, the expected delivered output of fixed-rate approach is half of the baseline. Similarly for the proposed method, due to the restrict deadline, the sampling number is half of the expected maximum number. Fig. 10 shows the number of subscription processing. The proposed method produces slightly more outputs compared to the fixed-rate approach. The difference between real processing number and the expected number of the proposed method and fix-rated method are 18.9% and 32%, respectively. These errors might come from uneven streaming content and communication link fluctuation.

VI. Limitations and Discussion

We state and leave discussions in this section for two issues. The first issue is about the definition of response time in

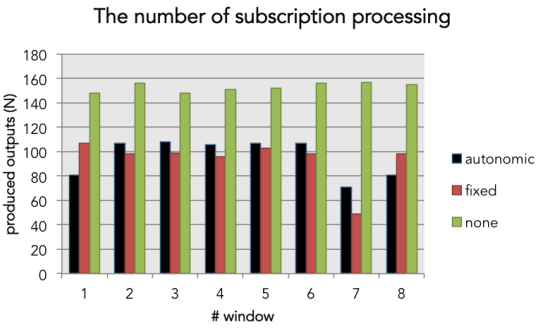


Fig. 10: The number of produced outputs for 8-windows run

sliding-window and aggregation processing [20]. The sliding window could be an application of *Reservoir* sampling which controls the number of reservoirs for each production. Either, it could cut off after a specific passing time. The latency in a waiting queue before processing may be ignorable. In this paper, we evaluate only on the simple processing flows. The second issue is about the accuracy of the learning model as well as the candidate selection algorithm in sampling number decision which could affect the efficiency of QoS control. The accuracy of the learning model depends on multiple factors including the number of learning items and the learning algorithm and features. At the current state, we use the sampling decision on each node as feedbacks but do not synchronize them to one solution. The synchronization mechanism might improve processing efficiency but must be a trade-off with communication overhead. We should further investigate the evaluation of these effects. To deal with dynamicity of networks, the more-frequent recalculation (computation window) might provide a better estimation. However, it must trade off with higher resource consumption.

VII. Conclusion

This paper introduces an autonomic sampling method for load shedding oriented by the deadline requirement aiming at coarse-grained stream processing requests. The deadline considers the response time, a time difference, computing from source to actuator in full-cycled stream processing system. Instead of controlling sampling probability or sampling rate, our proposed method set a goal at preserving the sampling number by adjusting the sampling rate during the operation window. The sampling number derives from the learning-based model and small-computation greedy function for picking up a candidate number. Also, we support the fully-distributed decision with a corresponding neighbor-feedback control mechanism. For evaluation, we implement sampling method overlaying our developed fully-distributed complex event processing engine, named EdgeCEP. The results show advantages of our method over fixed-rate sampling method in both terms of total latency (response time) and sampling controllability.

Acknowledgement

This work is supported in part by "Japan-US Network Opportunity 2 (JUNO2)", the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN, and JSPS KAKENHI 26220001.

References

[1] Q. Ni, L. Romdhani, and T. Turletti, "A survey of qos enhancements for ieee 802.11 wireless lan: Research articles," *Wireless Communications and Mobile Computing*, vol. 4, no. 5, pp. 547–566, 2004.

[2] A. Salama and R. Saatchi, "Adaptive sampling for qos traffic parameters using fuzzy system and regression model," 2017.

[3] C.-J. M. Liang, K. Chen, N. B. Priyantha, J. Liu, and F. Zhao, "Rushnet: Practical traffic prioritization for saturated wireless sensor networks," in *Proceedings of 12th Embedded Network Sensor Systems*, 2014, pp. 105–118.

[4] Q. Xiang, X. Chen, L. Kong, L. Rao, and X. Liu, "Data preference matters: A new perspective of safety data dissemination in vehicular ad hoc networks," in *Proceedings of 2015 Computer Communications*, 2015, pp. 1149–1157.

[5] D.-I. Sven Schmidt, "Quality-of-service-aware data stream processing," 11 2018.

[6] B. Babcock, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," in *Proceedings. 20th International Conference on Data Engineering*, April 2004, pp. 350–361.

[7] S. Choochotkaew, H. Yamaguchi, T. Higashino, M. Shibuya, and T. Hasegawa, "EdgeCEP: Fully-distributed Complex Event Processing on IoT Edges," in *Proceedings of The IEEE 13th International Conference on Distributed Computing in Sensor Systems (DCOSS2017)*, June 2017, pp. 121–129.

[8] S. Choochotkaew, H. Yamaguchi, and T. Higashino, "A self-organized task distribution framework for module-based event stream processing," *IEEE Access*, (to appear).

[9] S. Choochotkaew, H. Yamaguchi, T. Higashino, D. Schäfer, J. Edinger, and C. Becker, "Self-adaptive resource allocation for continuous task offloading in pervasive computing," *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 663–668, 2018.

[10] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surv.*, vol. 44, pp. 15:1–15:62, 2012.

[11] (2015) StreamBase. [Online]. Available: http://www.streambase.com

[12] "Storm: Distributed realtime computation system." [Online]. Available: http://storm.apache.org/

[13] G. Cugola and A. Margara, "Tesla: A formally defined event specification language," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. NY, USA: ACM, 2010, pp. 50–61.

[14] "WSO2." [Online]. Available: http://wso2.com/

[15] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets: "better than best-effort" computing," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2016, pp. 1–11.

[16] S. Choochotkaew, H. Yamaguchi, and T. Higashino, "Two-tier voi prioritization system on requirement-based data streaming toward iot," *Mobile Information Systems*, vol. 2017, pp. 1–16, 08 2017.

[17] P. J. Haas, *Data-Stream Sampling: Basic Techniques and Results*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 13–44.

[18] N. Tatbul, U. etintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *Proceedings 2003 VLDB Conference*, J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 309 – 320. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780127224428500355

[19] A. Salama and R. Saatchi, "Adaptive sampling for qos traffic parameters using fuzzy system and regression model," 2017.

[20] M. Hirzel, S. Schneider, and K. Tangwongsan, "Sliding-window aggregation algorithms: Tutorial," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '17. New York, NY, USA: ACM, 2017, pp. 11–14. [Online]. Available: http://doi.acm.org/10.1145/3093742.3095107