Poster: On Evaluating the Impact of Changes in IoT Data Streams Rate over Query Window Configurations

Radhya Sahal
National University of Ireland Galway
Galway, Ireland
radhya.sahal@nuigalway.ie

John G. Breslin
National University of Ireland Galway
Galway, Ireland
john.breslin@nuigalway.ie

Muhammad Intizar Ali
National University of Ireland Galway
Galway, Ireland
ali.intizar@nuigalway.ie

ABSTRACT

With the ever increasing number of IoT devices getting connected, an enormous amount of streaming data is being produced with very high velocity. In order to process these large number of data streams, a variety of stream processing platforms and query engines are emerging. In the stream query processing, an infinite data stream is divided into small chunks of finite data using a window operator. Window size and its type play an important role in the performance of any stream query engine. Due to the dynamic nature of IoT, data stream rate fluctuates very often, thus impeding the performance of query engines. In this work, we investigated the impact of any changes in data stream rates over the performance of a distributed query engine (e.g. Flink - https://flink.apache.org/). Our evaluation results indicate a direct impact of any changes in stream rate and window size over the performance of the engines. We propose an adaptive and dynamic query window size and type selector to improve the resilience of query processing engines. We consider several characteristics of input data streams, application workload, and resource constraints and proposes an optimal stream query window size and type for stream query execution.

CCS CONCEPTS

• **[Database Management]: → System-Query Processing**.

KEYWORDS

IoT, Stream Processing, Query Optimization

ACM Reference Format:
Radhya Sahal, John G. Breslin, and Muhammad Intizar Ali. 2019. Poster: On Evaluating the Impact of Changes in IoT Data Streams Rate over Query Window Configurations. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3328905.3332509

1 IMPACT OF CHANGING STREAM RATE AND QUERY WINDOW

Stream query processing is resource extensive due to the continuous execution of queries over infinite data streams. In stream processing for each time window, there is one execution of a query. Larger window size requires more memory consumption to store intermediate results while a smaller window size requires more query executions. Building an optimal query plan is challenging due to the unpredictable and fluctuating rates of input data streams [1, 2].

In this study, we evaluated the impact of any changes in stream rate and window configurations over the performance of a stream processing engine. Consider a scenario of fire alarm detection query which processes two input data streams and observes their values at a fix interval for any fire hazard detection. Input streams/sensors are configured in such a way that they change their stream rate with the changing values (e.g. if temperature rises beyond a certain threshold, data observation rate is increased to ensure early detection of any fire event). A fix window size will either impede the performance of query engine due to large number of intermediate results or miss/delay the detection of fire event due to the larger window size. We conducted our experiments using two input data streams which are streamed using Kafka (https://kafka.apache.org/)and processed by Flink. Query results are stored in MongoDB as data sinks. We evaluated three types of windows supported by Flink, namely tumbling, sliding, and session. For the first input stream, we varied it to 10 different stream rates i.e., 60, 120, 180, 240, 300, 360, 420, 480, 540, and 600 tuples per minute. For the second input stream, we only produced one tuple per minute at a fixed rate. We assess the latency (time consumed between the input arrival and output generation) using two window sizes; (1) the small window size which is one minute denoted by 1-min, 2) the large window size is five minutes denoted by 5-min. To consider the queue delay for Kafka, we calculated the latency of window-based join queries executed on Flink as (1) Flink with Kafka (i.e., the time consumed between the event observed until the output is generated) and (2) Flink without Kafka (i.e., the time consumed between the event is already ingested into Flink till the output is generated) which are denoted by Flink w/Kafka and Flink wo/Kafka respectively.

As shown in Figure 1 and Figure 2, Flink w/Kafka and Flink wo/Kafka latency of queries increases linearly with the increase in stream rate for both 1-min and 5-min window sizes for three window types. In particular, the Flink w/Kafka grows fast due to Kafka delay of launched Flink Kafka consumers to consume the input streams and Flink Kafka producers to send back the output stream to Kafka. However, the average latency of Flink w/Kafka over 5-min window is longer than the 1-min because the ingested tuples (i.e., from the first minute to the fifth minute) wait until the Flink window triggers its output. The larger window size can result in large data size ingested into Flink, and when this large data is processed it incurs higher processing time/latency. Our evaluation confirms the impact of dynamic stream rate or other factors over

ACM ISBN 978-1-4503-6794-3/19/06…$15.00
https://doi.org/10.1145/3328905.3332509

the performance of the query engine. However, there is no one window size and/or type fitting all input stream rates. Our evaluation strengthens the case for the need for adaptive window type and size recommendation based on the input stream rate variation.

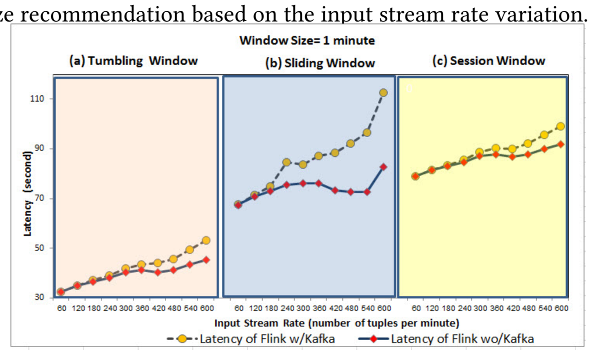

Figure 1: The latency of Flink join using window size 1-min.

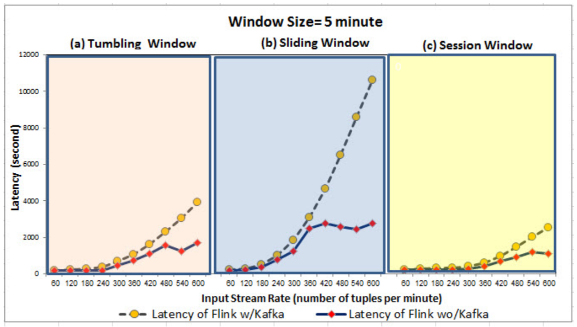

Figure 2: The latency of Flink join using window size 5-min.

2 DYNAMIC QUERY WINDOW CONFIGURATIONS

In this section, we describe our prototype for dynamic query window configurations depending on the changes in stream rate while considering stream attributes, workload requirements, and infrastructure specifications. Our typical approach is to deploy the query with an initial window configuration and then the query optimizer configures the optimal query window with respect to the identified inputs. Behavior of streams and their rates is continuously monitored to dynamically trigger deployment of any new optimal query window configurations. The proposed prototype contains the following three phases (see Figure 3):

Inputs Identification Phase. In this phase, the relevant inputs are identified including the stream attributes such as stream schema and historical stream statistics, the query workload requirements such as latency and accuracy, and the infrastructure specifications.

Window Selection Phase. This phase contains the following three components; 1) stream monitor 2) knowledge base and 3) query optimizer. A stream monitor includes a monitoring system to observe different characteristics of the input stream, A knowledge base contains the previously recommended window configurations for different stream rates including window type and size and their relation to the identified inputs. We built the knowledge base/heuristics to get a recommendation for window configuration with respect to workload requirements including latency and data enrichment over stream rate changes. The query optimizer contains cost estimator and enumerator module. The cost estimator is implemented as a cost model using different window configurations and identified parameters such as historical statistics, application requirements, resources constraints, and real-time analysis. The enumerator selects the window configuration with a minimum cost which should meet the application requirements while maintaining the resources constrains.

Deployment Phase. In this phase, the new optimized stream query will be deployed beside the current stream query (i.e., without killing the running query) and the current query still runs until the new one is warmed up.

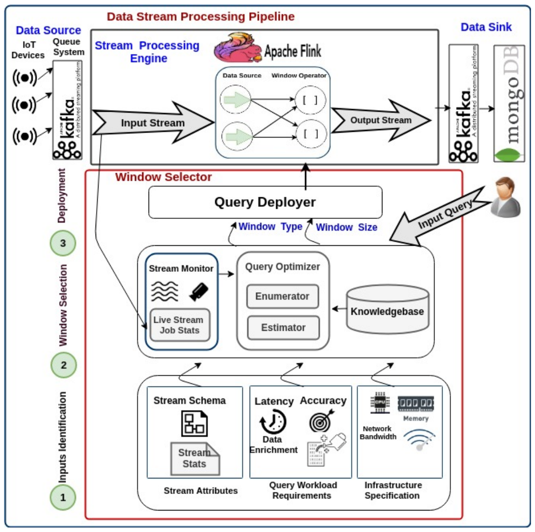

Figure 3: The prototype of a dynamic window-based selector.

3 CONCLUSION

In this work, we investigated the performance of stream query engines with respect to the dynamic external factors such as input stream rate, application requirements, device limitations, and resource constraints. Our evaluation results build a strong case for the need of adaptive stream processing techniques to effectively handle IoT data streams. In the future, we plan to further extend our work and augment our approach within top distributed stream processing platforms. Additionally, we also intend to build shared knowledge base containing statistics of the performance of IoT devices and their stream rates, which can be utilized by multiple platforms and applications to build robust and adaptive IoT applications.

Acknowledgement

This work is partially funded by SFI under grant no. SFI/16/RC/3918 and SFI/12/RC/2289.

REFERENCES

[1] João Gama and Mohamed Medhat Gaber. 2007. *Learning from data streams: processing techniques in sensor networks.* Springer.
[2] Christoph Hochreiner, Michael Vögler, Stefan Schulte, and Schahram Dustdar. 2016. Elastic stream processing for the internet of things. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on.* IEEE, 100–107.