Poster: Dependency-Aware Operator Placement of Distributed Stream Processing IoT Applications Deployed at the Edge

Alireza Mohtadi
Department of Software and IT Engineering
ÉTS Montréal / University of Quebec
Montreal, Quebec, Canada
alireza.mohtadi.1@ens.etsmtl.ca

Julien Gascon-Samson
Department of Software and IT Engineering
ÉTS Montréal / University of Quebec
Montreal, Quebec, Canada
julien.gascon-samson@etsmtl.ca

***Abstract*—In the last few years, the number of IoT applications that rely on stream processing has increased significantly. These applications process continuous streams of data with a low delay and provide valuable information. To meet the stringent latency requirements and the need for real-time results that they require, the components of the stream processing pipeline can be deployed directly onto the edge layer to benefit from the resources and capabilities that the swarm of edge devices can provide. In this poster, we outline some ongoing research ideas into deploying stream processing operators onto edge nodes, with the goal of minimizing latency while ensuring that the constraints of the devices and their network capabilities are respected. More precisely, we provide a modeling of the semantics of the operators that considers the interactions between different operators, the parallelism of concurrent operators, as well as the latency and bandwidth usage.**

***Index Terms*—Edge computing, Stream Processing, Operator Placement**

I. Introduction

IoT devices are becoming increasingly popular. They produce a huge amount of data that IoT applications can quickly process and consume to provide valuable information. In many contexts (e.g., smart cities), IoT devices produce continuous *streams* of data that must be processed with low delay [1].

In the traditional cloud-based model, the application is deployed on resources that are provided in a centralized manner [2]. In a stream processing context, continuous streams of data need to be sent to the cloud server, thus potentially incurring high bandwidth and latency. Given that most IoT applications are time-sensitive, deploying stream processing operators in a distributed manner *at the edge* (i.e., near the devices that produce and consume the data, or directly onto these devices) can help in reducing bandwidth usage and latency.

Different distributed stream processing frameworks have been proposed (e.g., Apache Storm [3], Twitter Heron [4], Apache Flink [5], etc), and most of them represent a given application as a directed acyclic graph (DAG). They use the data flow graph scheme and consider the operator as a black box that processes the data [6]. In a DAG, each edge represents an operator, and vertices represent the paths of data flows between operators. In a distributed stream processing (DSP) context, the operators are deployed on different nodes (e.g., on different *cloud*, or *edge* nodes in our case). These nodes provide the required resources for the operators (e.g., bandwidth, memory, processing power).

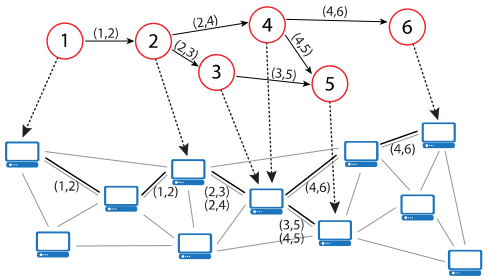

Fig. 1. Operator Placement on the edge nodes [7]

The goal of our research is to find an optimal mapping of operators to edge nodes (e.g., Figure 1), *while* respecting the constraints of the edge devices and the edge network topology. We target minimizing the latency as our optimization goal, given that low latency is a requirement of many IoT applications. We focus on bandwidth constraints, given that edge devices can have limited bandwidth and/or connectivity.

Existing works have considered different general scheduling algorithms without considering latency for each specific operator in placement problems. Other works have focused mostly on minimizing the latency on the specific path that has the highest latency (i.e., the critical path) – they have not considered the interaction between all the operators. Prior work has also looked at optimizing the sum of latencies across all edge nodes *while* considering the interactions between the operators [8]; however, the parallel execution of operators is not considered. Given that more than one operator can be run on a given edge node; therefore, the sum of the latencies of simultaneous operators cannot provide a good model for the objective function. Thus, in our research, we consider both the interactions between the operators and the parallel execution of these operators in the edge nodes.

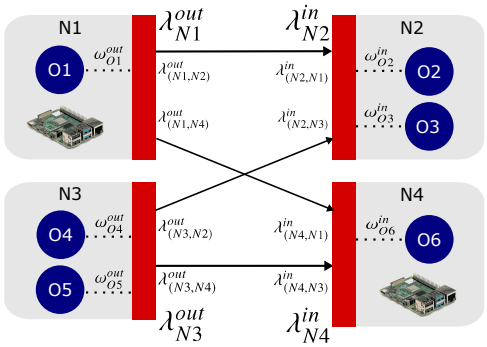

Fig. 2. Operator placement depends on the available edge network resources

TABLE I
PARAMETERS

Notation	Description
$G = (N, O)$	Problem graph
N_i	Edge node i
O_i	Operator i
B_i^{in}	Incoming bandwidth in node i
B_i^{out}	Outgoing bandwidth in node i
λ_i^{in}	Incoming throughput in node i
λ_i^{out}	Outgoing throughput in node i
$\lambda_{(i,j)}^{in}$	Incoming throughput from node j to node i
$\lambda_{(i,j)}^{out}$	Outgoing throughput from node i to node j
ω_i^{in}	Input data rate in operator i
ω_i^{out}	Output data rate in operator i
$FT_{(i,k)}$	Finish time of operator i placed on node k
E_i	Minimum number of messages that should be processed in operator i to produce the output
$T_{(i,k)}$	Time required to run operator i on node k (per message)

II. System Model and Problem Formulation

We model our problem as a DAG. Figure 2 and Table I describe some of the parameters that we used in our modeling. We consider different operator types, according to the classification in [9]. The diversity of the operator types led us to consider more features, such as the data dependency of operators, which can significantly impact the latency, as it affects the time at which a given set of operators will be able to execute. We consider data-dependent operators, which must wait for the data from upstream operators before processing. For instance, some operators (e.g., aggregation, join) generate their output by applying a transformation over multiple incoming data items (e.g., computing a moving average). Alternatively, data-independent operators can fire whenever any message is received. Figure 3 shows two sample DAGs with the data-dependent and data-independent semantics.

III. Objective Function

The objective of our operator placement problem is to minimize the total latency considering end-to-end latency between operators while meeting the bandwidth constraints. At a high level, our model allows us to determine the *starting time* (before processing) and *ending time* (after processing) of a given operator when deployed onto a specific node, which then allows us to infer the *finish time* of the *last* operator.

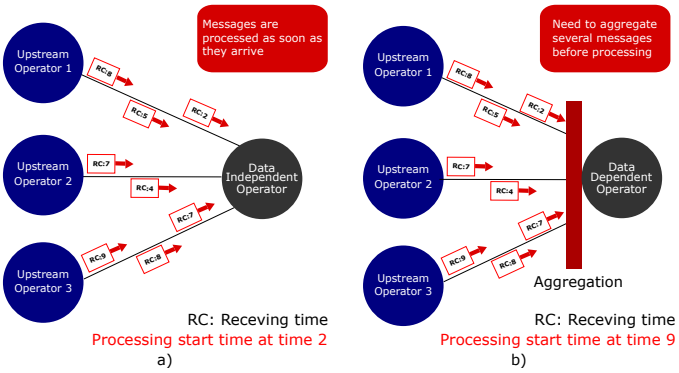

Fig. 3. a) Incoming messages to a data-independent operator b) Incoming messages to a data-dependent operator

For example, for data-dependent operators (equations 1, 2 and 3), we need to wait and start processing after receiving all required messages. Therefore, the start time of processing the first message $StartTime_1$ is equal to the last message receiving time from upstream operators. In the equation 2 and 3, variable b indicates a specific message of an operator.

$$FT_{(i,k)} = EndTime_{E_i} \quad \forall i \in O; k \in N \tag{1}$$

where:

$$EndTime_b = StartTime_b + T_{(i,k)} \quad 1 \leq b \leq E_i; \tag{2}$$

$$StartTime_b = EndTime_{b-1} \quad 2 \leq b \leq E_i; \tag{3}$$

IV. Constraints

Our model currently considers the bandwidth of the edge nodes (incoming and outgoing) as the main constraint in our problem. At a high level, the *sum of the required incoming and outgoing bandwidth for each operator* (which depend on the characteristics of the operator), deployed onto a given node, must not exceed the *available incoming and outgoing bandwidth available for that node* (equations 4 and 5). Integrating other constraints is an area of future work.

$$\lambda_k^{out} \leq B_k^{out} \quad \forall k \in N \tag{4}$$

$$\lambda_k^{in} \leq B_k^{in} \quad \forall k \in N \tag{5}$$

V. Evaluation Strategy

We plan to implement the proposed approach on a testbed of IoT devices (e.g., various flavors of Raspberry Pis) and edge networks, over Apache Storm [3]. We will emulate network conditions using a tool such as Netem [6], and we will compare our results with the prior approaches [8], [2].

VI. Conclusion

This paper presents a high-level overview of our current research on deploying distributed stream processing operators at the edge. We aim at reducing the latency in stream processing and IoT applications while meeting the bandwidth constraints of the edge devices and networks. In future work, we plan to work on the scheduling algorithms that can improve the latency, considering the new proposed objective function.

REFERENCES

[1] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[2] A. da Silva Veith, M. D. de Assuncao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 215–229.

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

[4] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 239–250.

[5] Apache Flink. [Online]. Available: https://flink.apache.org

[6] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator replication and placement for distributed stream processing systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 11–22, 2017.

[7] M. Nardelli, V. Cardellini, V. Grassi, and F. L. Presti, "Efficient operator placement for distributed data stream processing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1753–1767, 2019.

[8] G. Amarasinghe, M. D. de Assunçao, A. Harwood, and S. Karunasekera, "A data stream processing optimisation framework for edge computing applications," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 91–98.

[9] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An iot benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.