Rethinking Operators Placement of Stream Data Application in the Edge

Thomas Lambert, David Guyon, Shadi Ibrahim

Rethinking Operators Placement of Stream Data Application in the Edge

Thomas Lambert
Inria, IMT Atlantique, LS2N

David Guyon
Inria, IMT Atlantique, LS2N

Shadi Ibrahim
Inria, IMT Atlantique, LS2N

ABSTRACT

Maximum Sustainable Throughput (MST) refers to the amount of data that a Data Stream Processing (DSP) system can ingest while keeping stable performance. It has been acknowledged as an accurate metric to evaluate the performance of stream data processing. Yet, existing operators placements continue to focus on latency and throughput, not MST, as main performance objective when deploying stream data applications in the Edge. In this paper, we argue that MST should be used as an optimization objective when placing operators. This is specially important in the Edge, where network bandwidth and data streams are highly dynamic. We demonstrate that through the design and evaluation of a MST-driven operators placement (based on constraint programming) for stream data applications. Through simulations, we show how existing placement strategies that target overall communications reduction often fail to keep up with the rate of data streams. Importantly, the constraint programming-based operators placement is able to sustain up to 5x increased data ingestion compared to baseline strategies.

KEYWORDS

Data Stream Processing, Edge, Maximum Sustainable Throughput, Operators Placement

1 INTRODUCTION

The mutual low-latency objective for both Data Stream Processing (DSP) and Edge environments has resulted in a continuous growth of DSP deployments on Edge or Fog environments [11]. The success of DSP deployments in the Edge relies on operators placements and the ability to sustain low latency. Accordingly, much work have focused on placement strategies across Edge-servers [10] or across hybrid Cloud and Edge environments [4]. All of these studies have employed the same metrics and optimization objectives as previous efforts that handle resource heterogeneity in the Cloud [12]. Specifically, they target reducing latency or response time (sometimes included in a Quality of Service metric) [10] or minimizing the overall volume of data exchanged between nodes in the Edge [5].

The input rates of data streams do not usually remain unchanged. For example, the input rate of messages from two million trips taken in 2013 on New York City taxies ranges from 300 to 5200 msg/s [13]. When the size of input data grows, previous placements – where network heterogeneity of the Edge is often considered to minimize delays – can not scale, especially as Edge features with limited resources. The amount of data that a DSP system can ingest while keeping stable performance is referred to as Maximum Sustainable Throughput (MST). Existing works in literature have acknowledged the importance of MST as an accurate metric to evaluate the performance of DSP systems [6, 7].

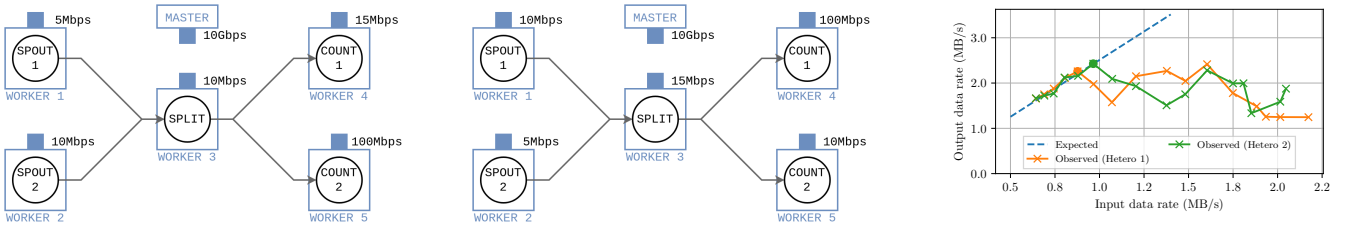
Given the dynamic nature of data streams (i.e., data volatility and bursts), we argue that MST should be considered as an optimization objective for operators placements in the Edge. Accordingly, we propose a model to evaluate MST for operators placement in the Edge, with a strong focus on the heterogeneous nature of network. We then introduce an optimal operators placement for stream data applications in the Edge using constraint programming. Through simulations, we show how existing placement strategies that target overall communications reduction often fail to keep up with the rate of data streams. Specifically, we demonstrate that the constraint programming-based operators placement is able to sustain up to 5x increased data ingestion compared to resource-aware placements (i.e., R-Strom [12]) and graph partitioning-based placement [3]. It is important to note that we are interested more precisely in long-running applications (where sustainability is crucial) and thus our model is designed to target the stable state (after the end of deployment) of such applications.

The rest of this paper is organized as follows. In Section 2, we analyze the impact of network heterogeneity on the Maximum Sustainable Throughput of stream data applications. Section 3 introduces our model and a constraint programming formulation for linear applications. We evaluate our bandwidth-aware placement using simulation in Section 4 and conclude this paper in Section 5.

2 MST: OCCURRENCE AND IMPLICATIONS

To assess the impact of network heterogeneity and data stream volatility on the performance instability of stream data applications in the Edge, we conduct a set of experiments using the Apache Storm DSP engine (version 2.1.0) [1]. In Storm, an application is called a *topology*. It is composed of spout and bolt operators that respectively emit and process data tuples.

Experimental setup. Our experiments are conducted on the French scientific testbed Grid'5000 [2]. We use 6 nodes from Ecotype cluster at the site of Nantes. We design two close scenarios with 1 Master and 5 Worker nodes. The goal of these scenarios is to show the performance differences when deploying the same topology (i.e., application) on two different heterogeneous network environments. We vary bandwidths from 50Mbps to 200Mbps (using the `tc tbf` command) to represent low-capacity Edge links and we inject network traffic (with `iperf3` using the UDP protocol) to represent network workloads. The set-up for each bandwidth is represented on Figures 1a and 1b. It is important to remind that as we focus on communications, we ensure that CPUs and memory are not bottlenecks. The considered topology is the wordcount example provided within the *storm-starter* package. This topology is composed of 3 operators: a *sentence* spout emitting a sentence at a fixed rate (experimental variable), a *split* bolt splitting sentences into words, and a *count* bolt that counts the occurrence of each word streamed so far. Default average size of each sentence (29.6B) is increased to an average of 96KB to represent a heavier workload. Within this topology, we vary the waiting time between emitting



(a) First experimental set-up (Hetero 1).

(b) Second experimental set-up (Hetero 2).

(c) Throughput of wordcount application when increasing the input data rate.

Figure 1: Impact of network heterogeneity and data stream volatility on the performance of wordcount application.

sentences (from spout) in order to control the application workload. Note that the amount of data emitted by the *split* bolt is 0.959 times the amount of data received (small decrease caused by white space suppression) and the amount of data written by *count* bolts is 1.309 times the amount of data received (augmentation caused by the addition of an integer).

Results and findings. We present the results in Figure 1c. The x-axis is the input data rate, i.e., the amount of data emitted by the spout per second. The y-axis is the output data rate, i.e., the amount of data written to final files by the *count* bolts. Experiments run for a duration of 6 minutes, but the first minute (where deployment occurs) is not included as we are focusing on the stable state (more representative for the behavior of long-run applications). We draw a theoretical line that represents the expected scaling (input rate $\times 0.959 \times 1.309$, the multiplication factors of *split* and *count*). From Figure 1c, we can draw the two following observations. First, there is a point where performance stops following the expected throughput. This happens when the input data rate increases, and network is no longer able to handle the amount of data sent through it. This inflexion point is known as Maximum Sustainable Throughout (MST) [6, 7]. It is clear that after this point the performances are unstable. Second, the heterogeneity causes differences. In particular, the MST for wordcount application is 0.88 MB/s in the Hetero 1 set-up (shown in Figure 1a), while it is 0.96 MB/s in the Hetero 2 set-up (shown in Figure 1b).

Implications. It is clear that to ensure stable performance when deploying stream data applications in the Edge, we need to improve the MST of operators placements. Thus, we need to target MST as an optimization objective for operators placement. This is acknowledged by previous work [7] which demonstrates that MST can accurately capture the performance of stream data applications (including latency). The second observation leads us to the conclusion that there is a need for heterogeneity-aware placements when deploying DSP systems in the Edge and when dealing with this scalability metric (MST).

3 MODEL

In this section, we propose a simple model which aims to represent the bandwidth consumption of an operators placement and how we deduce from it an estimation for the MST. For the sake of simplification, we focus on communications, assuming that CPU or memory are not bottlenecks (operators are sufficiently replicated).

3.1 Preliminary Definitions

Application. First let $G_O = (V_O, E_O, w_O)$ be a weighted DAG that represents the application with its operators and their dependencies. V_O is the set of operators (two instances of the same operator are two different vertices of V_O), E_O is the set of edges ($oo' \in E_O$ if and only if o sends data to o') and $w_O : E_O \to \mathbb{R}$ is the weight function that represents the amount of data sent by one operator to another.

Platform. We represent the targeted platform as a graph $G_N = (V_N, E_N)$. Each node $v \in V_N$ has a slot number s_v and each edge $uv \in E_N$ has a maximum bandwidth b_{uv}. We assume the graph to be complete and we also assume up and down bandwidths to be equal (edges are not directed).

Placement. A placement is a function $\sigma : V_O \to V_N$ such that for all $v \in V_N$, $|\sigma^{-1}(v)| \leq s_v$ (the number of operators allocated to each node is smaller or equal to its number of slots).

We define the *bandwidth usage* (*BU*) from node u to node v for a given placement σ as $BU(\sigma, u, v) = \sum_{oo' \in E_O, \sigma(o)=u, \sigma(o')=v} w_O(oo')$.

Finally, we define *bandwidth exceeding* (*BE*) as the overhead between bandwidth usage and the maximum bandwidth. More precisely, $BE(\sigma, u, v) = \max(0, BU(\sigma, u, v) - b_{uv})$ and total bandwidth exceeding is defined as $BE(\sigma) = \sum_{u,v \in V_N^2} BE(\sigma, u, v)$.

3.2 Maximum Sustainable Throughput

Let $G_O = (V_O, E_O, w_O)$ be a weighted DAG that represents the application with its operators and their dependencies, as described in Section 3.1. We now assume that $w_O(oo') = \beta_{oo'} \times D$ where D is the input data rate and $\beta_{oo'}$ is a fixed multiplication factor that is given for all edges in E_O (we assume that the evolution of all communications is linear when input rate changes). Let $G_N = (V_N, E_N)$ be a network topology and σ be a valid placement. We define the Maximal Sustainable Throughput (MST) τ as the maximal input rate D such that $BE(\sigma) = 0$.

More precisely (as $w_O(oo') = \beta_{oo'} \times D$), for each $u, v \in V_N^2$, $BE(\sigma, u, v) = 0$ if and only if $\sum_{oo' \in E_O, \sigma(o)=u, \sigma(o')=v} \beta_{oo'} \times D \leq b_{uv}$, which is equivalent to $D \leq \frac{b_{uv}}{\sum_{oo' \in E_O, \sigma(o)=u, \sigma(o')=v} \beta_{oo'}}$.

Hence $\tau = \min_{u,v \in V_N^2} \frac{b_{uv}}{\sum_{oo' \in E_O, \sigma(o)=u, \sigma(o')=v} \beta_{oo'}}$.

From this definition we now define the optimization problem we want to solve.

Problem 1 (MST-Maximization). *Let $G_O = (V_O, E_O, w_O)$ be an application DAG and G_N be a network topology graph. Return a placement σ that maximizes τ.*

3.3 Constraint Programming Formulation for Linear Applications

A linear application is split into *stages*, each stage is composed of several replicas of the same operator. Each replica sends the same amount of data to each replica of the following stage and receives the same amount of data from each replica of the previous stage. Each stage S_i is associated with a data multiplier α_i. If we denote by In_i the amount of data received per unit of time by all the operators, then $In_1 = D$ and $In_{i+1} = \alpha_i \times In_i$ (if $\alpha_i < 1$, the operators send less data than what they receive, if $\alpha_i > 1$, they send more data than what they receive). We denote by r_i the replication level of each stage (how many operators in stage S_i) and by D the input data rate that is emitted by S_0. To transform such an application into a DAG $G_O = (V_O, E_O, w_O)$ as defined in Section 3.1, we simply set $V_O = \bigcup S_i$, $oo' \in E_O$ if and only if $o \in S_i$ and $o' \in S_{i+1}$ and $w(oo') = In_{i+1}/(r_i \times r_{i+1})$. We also define β_i as $\beta_0 = 1/r_1$ and $\beta_{i+1} = \beta_i \times \alpha_i \times (r_{i-1}/r_{i+1})$. This way, one can prove that if $o \in S_i$ and $o' \in S_{i+1}$, then $w(oo') = \beta_i D$ and thus $b_{oo'} = \beta_i$.

Constraint programming (CP) is generalization of linear programming where decision variables can be multiplied. The general problem is NP-complete. We here propose a transcription of MST-Maximization problem for linear applications into a CP model. We define for that one family of decision variables: $\forall i \in [0, N],\ \forall v \in V_N,\ x_{i,v}$ where $x_{i,v}$ represents the number of operators of stage S_i which are placed on the node u ($x_{i,v}$ is an integer). The constraint problem formulation of MST-Maximization problem for linear application is then:

Maximize D under

$$\forall i \in [0, N],\ \forall v \in V_N,\ x_{i,v} \geq 0 \tag{1}$$

$$\forall i \in [0, N],\ \sum_{v \in V_N} x_{i,v} = r_i \tag{2}$$

$$\forall v \in V_N,\ \sum_{i \in [0,N]} x_{i,v} \leq s_v \tag{3}$$

$$\forall u, v \in V_N,\ u \neq v,\ \sum_{i \in [0,N-1]} x_{i,u} x_{i+1,v} \beta_i D \leq b_{uv}. \tag{4}$$

Equation (1) ensures that $x_{i,v}$ is positive, Eq. (2) ensures that all operators of each stage have been allocated and Eq. (3) ensures that no node receives more operators than its number of slots. Finally, Eq. (4) evaluates the amount of data transferred from node u to node v and ensures it is lower than the available bandwidth between these two nodes. More precisely, if $x_{i,u}$ operators of S_i are allocated on node u and $x_{i+1,v}$ operators of S_{i+1} are allocated on node v, then $x_{i,u}x_{i+1,v}\beta_i D$ is the amount of data transferred between u and v between S_i and S_{i+1}, as there is $x_{i,u}x_{i+1,v}$ edge of E_O with weight $\beta_i D$.

4 EVALUATION

We develop a simulator to evaluate the MST of three baseline placement strategies – representatives of communication-oriented scheduling policies – and the optimal placements obtained through constraint programming model.

4.1 Placement Strategies

Greedy placements. We proposed here two variants of a greedy placement inspired by the R-Storm scheduler proposed by Peng et al. [12]. In R-Storm, G_O is first traversed following a BFS (Breadth-First-Search) order. The node with most available resources is selected as the referent node. The first operator is then allocated to this referent node and following operators are allocated to the nodes that meet best their requirements (in terms of CPU and memory) while being close enough to the referent node (the choice is made through a weighted distance function). To mimic this placement strategy in our model, we propose two variants. In both cases, operators are sorted by a BFS-traversal order and the node with the largest number of slots is assigned to operators (an operator to each available slot). Then, the next operators are given either to the second node with the largest number of slots (we denote this strategy as *GreedySlots*), or to the node that is the closest (i.e., with the highest bandwidth) to the referent node (we denote this strategy as *GreedyDistance*).

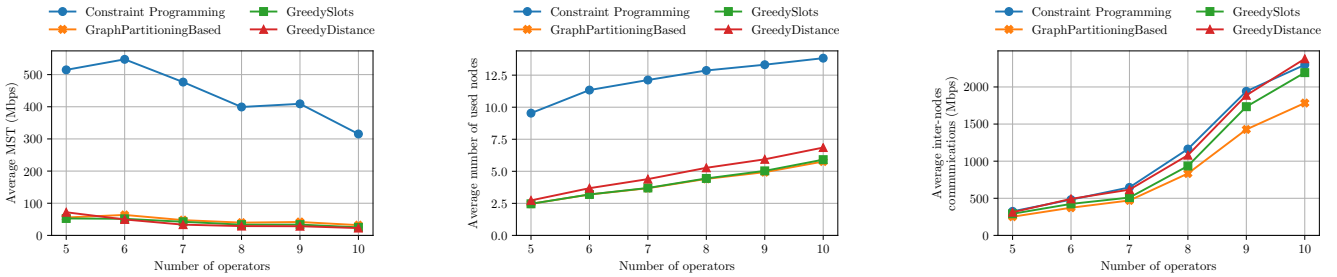
Graph partitioning-based placement. The strategy we propose here is based on the graph partitioning problem. Given graph, the goal is to partition the vertices of this graph into n sets of the same cardinality while minimizing the weight of the edge-cut, i.e., the sum of weights on the edges that goes from one set to another. More precisely, let $G = (V, E, w)$ be a graph with weight on edges. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a partition of V, i.e., $\bigcup P_i = V$ and $\forall i \neq j,\ P_i \cap P_j = \emptyset$. The edge-cut of this partition is then defined as $E_{cut} \subset E$, with $uv \in E_{cut}$ if and only if $u \in P_i,\ v \in P_j$ and $i \neq j$.

Graph partitioning is a very classical way to minimize communications between different processes. This technique has already been applied by Fischer and Bernstein to data stream processing [3] and we propose here to evaluate its efficiency with our model. As graph partitioning solver, we use the well-established METIS which can be applied to heterogeneous partitions (different size for each P_i) [8]. The placement strategy, denoted as *GraphPartitioningBased*, sorts the nodes by decreasing number of slots and chooses the k first ones with k being the smallest value such that $S_k = \sum_{1 \leq i \leq k} s_{v_i} \geq |V_O|$. $METIS(G_O, k, \{s_{v_1}/S_k, \ldots, s_{v_k}/S_k\})$ is then called to define the placement. Note that to improve the edge-cut of the resulted partitions, METIS does not always respect the initial load-balancing. To avoid non-valid placement, we add a small correction phase. If a node has more allocated operators than slots, then the operator with minimum data sent or received is moved to the first node with one free slot.

Constraint programming. We rely on the IBM solver (i.e., CP Optimizer [9]) to compute optimal results from the formulation introduced in Section 3.3. The resulted optimal placement targets maximizing MST while considering the bandwidth heterogeneity in the Edge.

4.2 Platform and Applications

Platform. We propose, as platform, an Edge environment with 15 nodes. For each node v, s_v is randomly chosen from 5 to 10. For each edge uv, the bandwidth is randomly chosen between 10 and 100 Mbps and are of the same magnitude as the ones used in [11] for Edge environments.



(a) Average Maximum Sustainable Throughput.

(b) Average number of nodes with at least one operator.

(c) Average volume (per second) of inter-nodes communications.

Figure 2: Comparison results of GreedySlots, GreedyDistance, GraphPartitioningBased and Constraint Programming.

Application. For this set of experiments, we generate random linear applications with 5 to 10 stages (which increase the number of operators and thus the amount of exchanges). r_i is randomly chosen from 2 to 10 (except first and last stages that only have one instance). α_i is either chosen in $[1/3, 1]$ or in $[1, 3]$ with equal chances (it has one chance over two of increasing data or one chance over two of reducing data). In total, we generate 100 applications. In all cases, the random distributions are uniform.

4.3 Results

Results for MST are shown in Figure 2a. We also present the number of used nodes (number of nodes with at least one operator) in Figure 2b and the volume (per second) of inter-node communications in Figure 2c to complete the analysis. We can clearly see that the baseline strategies result in low MST compared to the optimal. Constraint programming-based placement achieves up to 5x higher MST compared to the three baseline strategies (on average it can sustain an input data rate of 500 Mbps while the sustainable input data rates under other strategies are, on average, below 100 Mbps). We also note a reduction in MST when the number of stages increases, especially under constraint programming. One reason to explain these differences is that communication-oriented strategies try to group operators to maximize intra-node communications. This can result in low inter node-communication (lower or similar to the one obtained by constraint programming-based placement, as shown in Figure 2c), but as they are using fewer links to communicate data, some links can quickly reach their maximum capacities. This is why optimal solution spreads operators on more than 10 nodes on average (see Figure 2b). This supports our observation that targeting the volume of inter-node communications may not be a good metric when dealing with bandwidth heterogeneity.

5 CONCLUSION

In this paper, we propose a model to predict Maximum Sustainable Throughput of stream data applications in the Edge. Using this model, we show that communication-oriented strategies often fail to propose scalable placements, optimal one being often able to handle 5x more input data. This confirms that placement strategies for DSP in the Edge has to be bandwidth-aware. In future works, we plan to implement our constraint programming solution in Storm and evaluate it on a real testbed. We also plan to test the scalability limits of constraint programming approach (in terms of computation time) and work on the design of heterogeneity-aware heuristics when this computation time is no longer acceptable.

ACKNOWLEDGMENT

This work is supported by the ANR KerStream project (ANR-16-CE25-0014-01) and the Stack/Apollo connect talent project. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

REFERENCES

[1] Apache Storm. 2020. https://storm.apache.org/
[2] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez *et al.* 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*. Communications in Computer and Information Science, Vol. 367. 3–20.
[3] Lorenz Fischer and Abraham Bernstein. 2015. Workload scheduling in distributed stream processors using graph partitioning. In *IEEE Big Data '15*. 124–133.
[4] Eduard Gibert Renart, Alexandre da Silva Veith, Daniel Balouek-Thomert, Marcos Dias de Assuncao, Laurent Lefèvre, and Manish Parashar. 2019. Distributed Operator Placement for IoT Data Analytics Across Edge and Cloud Resources. In *IEEE/ACM CCGrid '19*. 459–468.
[5] Lin Gu, Deze Zeng, Song Guo, Yong Xiang, and Jiankun Hu. 2016. A general communication cost optimization framework for big data stream processing in geo-distributed data centers. *IEEE Trans. Comput.* 65, 1 (2016), 19–29.
[6] Shigeru Imai, Stacy Patterson, and Carlos A Varela. 2017. Maximum sustainable throughput prediction for data stream processing over public clouds. In *IEEE/ACM CCGrid '17*. 504–513.
[7] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking distributed stream data processing systems. In *IEEE ICDE '18*. 1507–1518.
[8] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.
[9] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. 2018. IBM ILOG CP optimizer for scheduling. *Constraints* 23, 2 (2018), 210–250.
[10] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco Lo Presti. 2019. Efficient Operator Placement for Distributed Data Stream Processing Applications. *IEEE TPDS* 30, 8 (2019), 1753–1767.
[11] Shadi Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. 2020. The Emerging Landscape of Edge-Computing. *ACM GetMobile: Mobile Comp. and Comm.* 23, 4 (2020), 11–20.
[12] Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell. 2015. R-storm: Resource-aware scheduling in storm. In *ACM Middleware '15*. 149–161.
[13] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. 2017. RIoTBench: An IoT benchmark for distributed stream processing systems. *Concurrency Computat: Pract Exper* 29, 21 (2017), e4257.