A Throughput Model for Data Stream Processing on Fog Computing

Felipe Rodrigo de Souza, Marcos Dias de Assuncao, Eddy Caron

A Throughput Model for Data Stream Processing on Fog Computing

Felipe Rodrigo de Souza, Marcos Dias de Assunção, Eddy Caron
Univ. Lyon, EnsL, UCBL, CNRS, Inria, LIP
LYON Cedex 07, France
{felipe-rodrigo.de-souza, marcos.dias.de.assuncao, eddy.caron}@ens-lyon.fr.

***Abstract*—Today's society faces an unprecedented deluge of data that requires processing and analysis. Data Stream Processing (DSP) applications are often employed to extract valuable information in a timely manner as they can handle data as it is generated. The typical approach for deploying these applications explores the Cloud computing paradigm, which has limitations when data sources are geographically distributed, hence introducing high latency and achieving low processing throughput. To address these problems, current work attempts to take the computation closer to the edges of the Internet, exploring Fog computing. The effective adoption of this approach is achieved with proper throughput modeling that accounts for characteristics of the DSP application and Fog infrastructure, including the location of devices, processing and bandwidth requirements of the application, as well as selectivity and parallelism level of operators. In this work, we propose a throughput model for DSP applications embracing these characteristics. Results show that the model estimates the application throughput with less than 1% error.**

***Index Terms*—Data Stream Processing, Model, Evaluation, Throughput, Fog**

I. Introduction

The maturity of several IT technologies such as Cloud computing and high-end devices and sensors has brought our society to a *big data* era where data grows constantly at unprecedented levels in terms of volume, variety, and veracity [1], [2]. DSP applications have become a popular solution to cope with the volume and velocity of big data in a timely fashion. Such applications can process unbounded data streams in near real-time as data events are generated [1]. A DSP application is often described as a directed graph, with nodes representing operators, data sources and data sinks (data consumers); and edges referring to the data flows between operators. An operator is a processing unit that receives a continuous incoming stream, applies operations over it and generates a new output stream [1].

The Cloud has been the preferred infrastructure for deploying DSP applications due to its elastic capabilities and the virtually unlimited number of resources that it can provide. This is an efficient approach for applications that process data that is either generated by systems hosted in the Cloud itselt – like web analytics where the data is collected from web applications – or for applications that tolerate the delays posed by transferring data to the cloud for processing. However, with advances on Internet of Things (IoT) and 5G technologies, there is an increasing number of devices geographically distributed at the edges of the network generating data that requires timely processing and services with very short response time. For these cases, Cloud deployment is an inefficient approach due to the high latency that it introduces [3]. Recent work aims to explore Fog computing to overcome this problem.

Fog computing is a paradigm that splits the network infrastructure into three layers as depicted in Fig. 1. The first layer comprises numerous geographically distributed devices (*e.g.*, sensors, IoT devices) often acting as data sources. They can be used for deploying DSP operators since they offer non-negligible compute power, though they are constrained in terms of memory, storage, and energy consumption when compared to Cloud resources. The second layer contains geographically distributed devices, without stringent energy constraints and with more memory, storage and processing capacity than the first layer. Routers, gateways, and micro datacenters are examples of second-layer devices. The third layer is the Cloud with thousands of high-end servers with fewer constraints [4].

As Fog computing aims to push computation to the edges of the network, it can be leveraged to offload DSP applications, fully or partially, from the Cloud to devices on the first and second layers of Fog computing, hence minimizing the effects of network latency and traffic congestion during data processing. However, the process of deciding which operators from the DSP application to offload and on which Fog resources to place them requires solving an issue known as the *operator placement problem*. This problem consists of finding a set of physical resources to deploy operators while respecting the application requirements [5]. The application requirements are often defined by a user who has little or no knowledge on how to express them, mainly specifying the processing capacity, bandwidth, and parallelism level of operators [6]. *Throughput* is a metric commonly used to measure the performance of the application and quality of a deployment. Failure on properly specifying the application requirements can lead to deployments that incur high costs, application bottlenecks and low throughput.

This work presents a throughput model for DSP applications that, based on a deployment plan for Fog computing provided by a placement solution, can estimate the application throughput with less than 1% error. The model is a building block for deployment frameworks capable of estimating the parallelism level of operators, processing and bandwidth requirements.

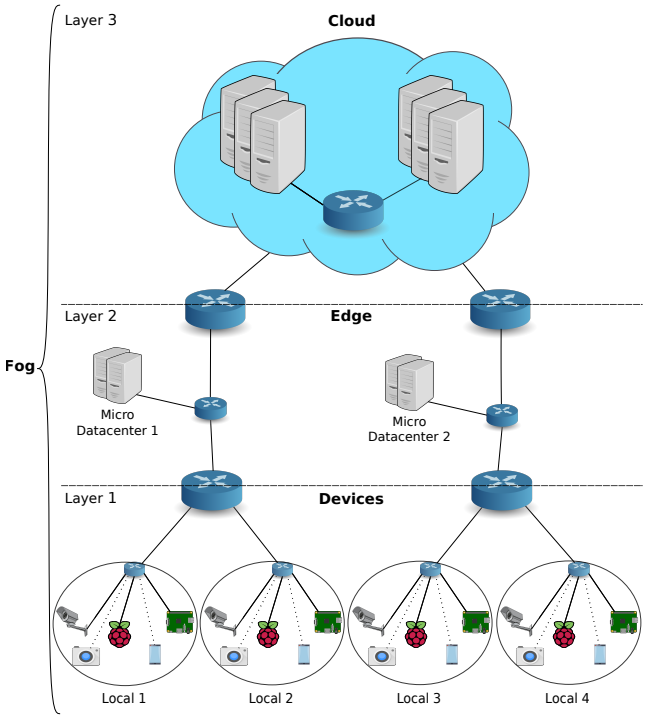

Fig. 1. Three-layered Fog infrastructure.

The remaining sections of this paper are as follows: Section II provides the envisioned architecture and background. In Section III the operator placement model and the proposed model are presented. Section V contains the setup information about the evaluation of the proposed model and the results. We discuss the related work in Section VI and finally Section VII concludes the paper and presents future directions.

II. Data Stream Processing Architecture

In this work, we propose a model to evaluate the throughput of deployment plans for DSP applications on Fog computing. The architecture to assist deployment solutions and ease the deployment process in highly distributed infrastructure is depicted in Fig. 2. The architecture comprises modules to manage and extract information from the physical resources and a deployment engine for DSP applications.

The physical resources follow the layered structure of Fog computing: IoT devices, micro datacenters, and Cloud computing. A *Node Manager* manages and deploys operators on a cloud or edge resource and is composed of two modules, namely the *Operator Management* and *Performance Monitoring*. The former manages and deploys operators whereas the second collects performance metrics from the device and reports it to the *Resource Manager* in the *Data Stream Processing Engine*, a centralized module responsible for managing the physical resources. The *Resource Manager* employs techniques such as Vivaldi [7] or Software Defined Networking (SDN) [8] to discover the network topology and its status.

The deployment process begins with a user submitting a dataflow and its deployment requirements to the *Data Stream Processing Engine* via a *Dataflow API*. This application submission contains the description of the application graph, the requirements and properties of each operator, and the deployment goal, which can be throughput maximization, latency minimization, cost minimization, or a combination thereof. The application and its requirements are then passed to the *Application Scheduler*, which along with information of the infrastructure, devises a *schedule/deployment plan*. Information on available resources, their residual capacity, their network interconnections, and the network capacity is provided by the *Resource Manager*.

The search for an application schedule or deployment plan can employ techniques that consider one or multiple criteria when placing DSP applications on a Fog environment. The scheduler should ideally include a model for each considered metric when aiming to optimize it. Finding a deployment plan for placing operators on the available resources is often termed as the *Operator Placement Problem*. In this context, the proposed *Throughput Model* is designed to work as a plug-in that during the search process provides a throughput evaluation for all evaluated deployment plans, thus resulting in a plan that maximizes the throughput.

After the *Application Scheduler* finds a solution, it provides the deployment plan with the description of devices where each operator should be placed, the requirements of each operator and the required network configuration. This deployment plan is submitted to the *Resource Manager* that coordinates with *Node Managers* and employs SDN techniques to enforce the network configuration. The DSP application can then start its execution.

III. System Model and Problem Statement

As mentioned earlier, the solution of the *operator placement problem* produces a *deployment plan*, which establishes where and how the operators should be deployed. This section introduces the model for estimating the throughput of a deployment plan for DSP applications on Fog computing. The notation used to describe the model is summarized in Table I.

The Fog computing infrastructure is composed of three device layers. The first layer comprises IoT devices, the second layer represents micro datacenters, and the third layer refers to the Cloud infrastructure. Hereafter these layers are called IoT, $Edge$, and $Cloud$ respectively. The infrastructure is represented by the graph $\mathcal{I} = \langle \langle D \cup H \rangle, C \rangle$, where $\langle D \cup H \rangle$ comprises all devices capable of deploying operators, with D representing the devices from the IoT layer and H the resources from $Edge$ and $Cloud$ layers; and C contains links interconnecting the devices inter and intra layers. The bandwidth between devices i and k is given by $Net(i,k)$.

A DSP application is a directed graph $\mathcal{A} = \langle O, E \rangle$ where O represents operators, data source(s) and data sink(s), and E refers to the streams between operators, which are any unbounded sequence of data (*e.g.*, messages, packets, tuples, file chunks) [9]. The application has at least one data source responsible for generating the input data stream that the operators consume. Each operator receives the data stream, applies

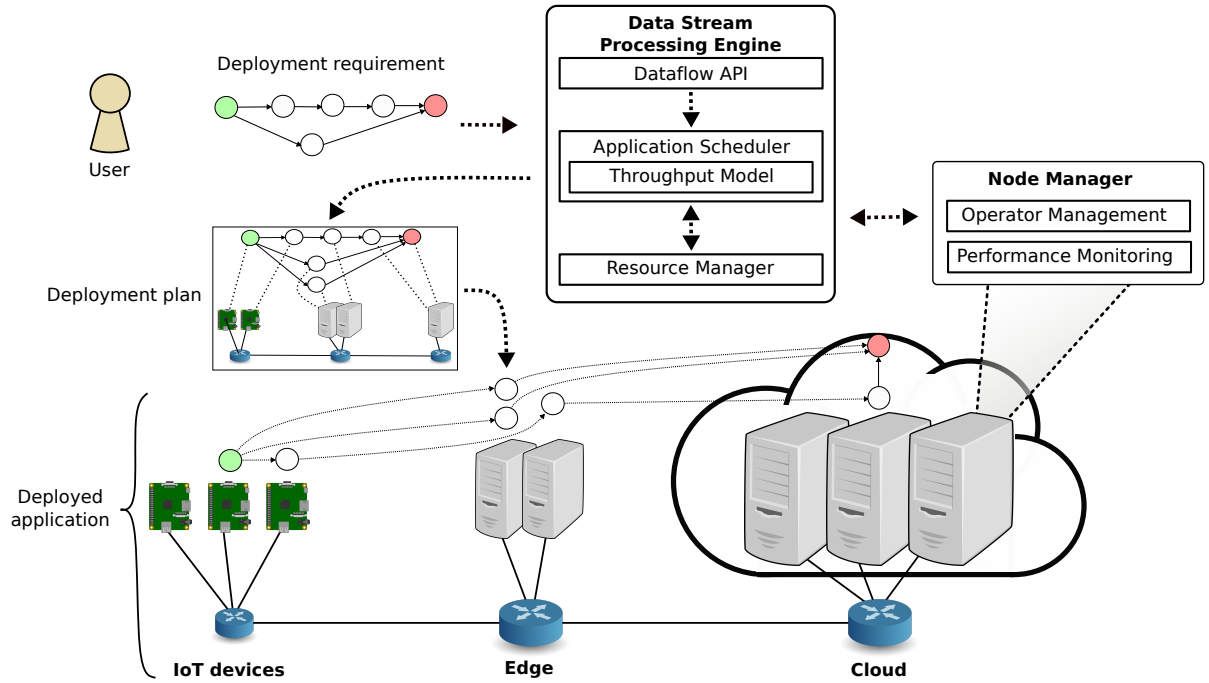

Fig. 2. System architecture.

TABLE I
NOTATION USED THROUGHOUT THE PAPER.

Notation	Description
D	Set of all IoT devices at layer 1
H	Set of all computing resources at layers 2 and 3
L	Set of network links interconnecting resources
$\mathcal{I} = \langle\langle D \cup H\rangle, L\rangle$	Fog infrastructure graph
l	Index of Fog layer (1, 2 or 3)
$Source(l)$	Set of source operators hosted in layer l
$Sink(l)$	Set of sink operators hosted in layer l
$\ell(j)$	Deployment layer of operator j
O	Set of all operators
$\mathcal{A} = \langle O, E\rangle$	Application graph with operators O and streams E
O_i	Set of all operators in device i
$M(j)$	Set of all devices hosting operator j
GR_i^j	Data rate received by $source$ operator j in IoT device i
AR_i^j	Arrival data rate for operator j hosted by device i
DR_i^j	Departure data rate for operator j hosted by device i
λ_i^j	Function that provides the arrival data rate of operators that are not application $source$
D^j	Demand in bytes per second of operator j
S^j	Selectivity rate of operator j
$\mathcal{P}(j\prime, j)$	Probability of operator $j\prime$ send data to j
$Rate(l)$	Data throughput of layer l
$PT(j)$	Processing time for $sink$ operator j
$PP(j)$	Higher processing time path for $sink$ operator j
$Prv(j)$	Set of all operators that send messages to j
$Net(i,k)$	Function that returns the bandwidth between devices i and k

a transformation on it (*e.g.*, filtering, projection, convolution) and produces another output stream. A stream flows between operators until it reaches a data sink. The application can have multiple data sinks where the data is stored or provided to a user or external system. An operator j is a processing unit composed of the tuple $\langle D_j, S_j\rangle$, where D_j is the processing requirements for the operator j, and selectivity S_j is the volume difference (in terms of number of messages and size of each message) in the data stream between the arrival and departure data rate.

The proposed model considers that a deployment plan follows the structure of the Fog computing infrastructure and splits the set of operators O into three deployment sequences, one for each layer l of the Fog (*i.e.*, IoT, $Edge$, $Cloud$) as depicted in Fig. 3. This division is used to compute the throughput in each layer and account for the network effect between layers. As it is essential to understand how the operators deployed in one layer interact with operators from other layers, we employ two functions $Sink(l)$ and $Source(l)$. The function $Source(l)$ provides the set of operators that receive data from operators in others layers, or from *data sources* in the IoT layer. $Sink(l)$ on the other hand provides all operators of layer l that stream data to operators in other layers, or to the application *data sinks* in the $Cloud$. The layer of an operator j is given by $\ell(j)$.

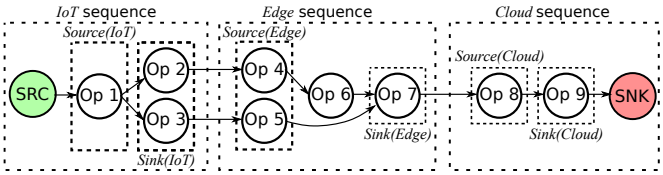

Fig. 3. Deployment sequences for each layer of Fog computing.

We assume that the deployment plan assigns the first

operator of an application to the sequence of the IoT layer, so trying to make the best use of the Fog infrastructure. The rationale behind this assumption comes from the fact that the first operator of the application receives data directly from the data source. To maximize the throughput it should be deployed as close as possible to the data source.

Therefore for operators in the subsequence $Source(IoT)$ the arrival rate AR_i^j (Eq. 1) is equal to the data generation rate GR_i^j in Bytes/s, of the number and size of generated messages. For the remaining operators, in any device $i \in \langle D \cup H \rangle$, the AR_i^j considers the parallelism level of operator j (*i.e.*, how many operator instances/tasks are executed) and the departure rate of all previous operators that send data to j ($Prv(j)$). Function $M(j)$ gives a map $\langle$*operator, device*$\rangle$ for all the deployed instances of operator j; thus the parallelism level of j is given by $|M(j)|$. The departure rate of all previous operators that send data to j is given by λ_i^j (Eq. 2), and if j and a previous operator j' are on the same device, simply the departure rate of j' is considered. Otherwise, it should take into account the departure rate of the devices deploying j' ($M(j')$). Moreover, since j' could send its output stream to various operators, the model considers the probability $\mathcal{P}(j', j)$ that j' will send the outgoing stream to j.

$$AR_i^j = \begin{cases} GR_i^j, & \text{if } j \in Source(IoT) \\ \frac{\lambda_i^j}{|M(j)|} & \text{otherwise} \end{cases} \quad (1)$$

$$\lambda_i^j = \sum_{j' \in Prv(j)} \sum_{k \in M(j')} \mathcal{S}_{i,k}^{j,j'} \quad (2)$$

$$\mathcal{S}_{i,k}^{j,j'} = \begin{cases} \mathcal{P}(j', j) \times DR_k^{j'} & \text{if } i = k \\ \min\left[\frac{\mathcal{P}(j', j) \times DR_k^{j'}}{Max(PT(j'), 1)}, Net(k, i)\right] & \text{otherwise} \end{cases} \quad (3)$$

The departure rate DR_i^j (Eq. 4) applies the selectivity S^j on the arrival rate, changing the number and size of messages in the income stream to produce the output stream. As a product of the arrival rate, the departure rate is also given in Bytes/s.

$$DR_i^j = AR_i^j \times (1 - S^j) \quad (4)$$

The application graph contains several paths between *sources* and *sinks*, and they can be decomposed as pipelines to be executed in parallel. To compute the throughput of each layer l the model applies the same notion, but considering the paths between $Source(l)$ and $Sink(l)$, where the throughput of each operator $j \in Sink(l)$ considers the path with high processing time from the $Source(l)$ to the operator itself. For instance, in the application in Fig. 3 the throughput of the *Edge* layer is based on the throughput of Op 7, and the paths to this operator are Op 4 $\rightarrow$ Op 6 $\rightarrow$ Op 7 and Op 5 $\rightarrow$ Op 7, then it is chosen the one with high processing time. In the model this process of identifying and selecting the path with high processing time, from the $Source(l)$ to $j \in Sink(l)$, is realized by function $PP(j)$.

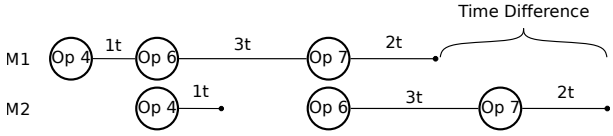

Fig. 4. Difference between the finish processing times of two consecutive messages.

After selecting the path with high processing time, the model computes the time difference between the departure timestamp of two consecutive messages, from the last operator on the path. Let us assume that for the throughput of the $Edge$ layer of the application in Fig. 3 the path with high processing time is Op 4 $\rightarrow$ Op 6 $\rightarrow$ Op 7, the pipeline execution of this path is depicted in Fig. 4. The message $M1$ arrives at Op 4 and is processed during 1 time unit, after that is handed to Op 6. While Op 6 starts to process $M1$, Op 4 starts to process $M2$. After 1 time unit Op 4 finishes $M2$ and stores it in a queue while Op 6 finishes processing $M1$. Both messages leave the path in Op 7, so we compute the time difference with the departure timestamp from this operator.

This time difference is given by $PT(j)$(Eq. 5), which is used to compute the throughput for operator j. The model computes this time difference for the arrival rate of the first operator of the path; therefore if this time difference is lower than one second, this means that operator j is able to process more than the arrival rate. However, since the throughput is the departure rate produced in one second, we consider one second as the minimum time difference between two consecutive messages.

$$PT(j) = \frac{AR_{M^{first(PP(j))}}^{first(PP(j))}}{D^{first(PP(j))}} + Q(j) - \sum_{k \in PP(j)} \frac{AR_{M^k}^k}{D^k} \quad (5)$$

where $Q(j)$ is given by:

$$Q(j) = \sum_{k \in PP(j)} \begin{cases} \frac{AR_{M^k}^k}{D^k} & k = last(PP(j)) \\ \max\left[\frac{AR_{M^k}^k}{D^k}, \frac{AR_{M^{k+1}}^{k+1}}{D^{k+1}}\right] & \text{otherwise} \end{cases} \quad (6)$$

The proposed model computes the throughput for each layer l (Eq. 7), based on the departure rate of all operators $j \in Sink(j)$ and the time difference $PT(j)$. Since the throughput is based on the departure rate, it is given in Bytes/s. The throughput of the DSP application is given by $Rate(Cloud)$.

$$Rate(l) = \sum_{j \in Sink(l)} \frac{DR_{M(j)}^j}{\max(PT(j), 1)} \quad (7)$$

IV. Experimental Setup

To evaluate the proposed model we use a real testbed as our Fog environment on which we deploy a DSP application.

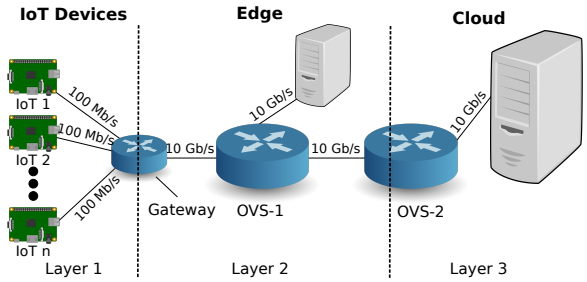

Fig. 5. Experimental Fog Infrastructure.

We compare the throughput obtained by the model against the throughput achieved by deploying the application on the testbed.

A. Infrastructure

The Fog computing testbed depicted in Fig. 5 is organized as follows. The *IoT* layer comprises 2 Raspberry PI's 3 (*i.e.*, ARMv7 at 1.2 GHz and 1 GB of RAM), both connected to a gateway via a 100Mb/s network and latency of 0.4ms [10]. The gateway is a server with an Intel® Xeon® E5-2620 at 2.10GHz and 64GB RAM, where the operators in the subsequence $Sink(IoT)$ of the *IoT* layer will publish their output stream to be read by the subsequence $Source(edge)$ on the *Edge* layer.

The *Edge* and *Cloud* layers contain four servers with an Intel® Xeon® X5550 at 2.67GHz with 32GB RAM and a NetFPGA card with four 10Gb/s ports. Two of those servers are used for the operator deployment, one in each layer (*Edge* and *Cloud*), and the two remaining servers are used as routers for the layers, running instances of Open vSwitch[1] 2.9 (OVS). The servers and their respective OVS are connected by 10Gb/s links, but due to limitations of the NetFPGA driver and from OVS the maximum bandwidth achieved is ≃2.3Gb/s. The latency between the gateway and the edge server is configured as ≃24ms and the latency between the edge server and the Cloud is ≃50ms [10].

B. Stream Processing Application

The proposed model estimates the throughput based on a deployment plan for a DSP application. In this experiment we use a sentiment analysis application [11], which evaluates the positive or negative sentiment associated with a tweet. The application structure is depicted in Fig. 6. The *data source* of the application is a data set with 50K tweets as JSON-format files with sizes ranging from 4 to 24 KB. The tweet data set is recursively read during the application execution. The *data sink* stores the data produced by the last operator.

The application is composed of a pipeline with five operators. Each operator is coupled with a queue that works as a buffer for the output data stream that will be read by the following operator in a First In First Out (FIFO) manner. The queues are implemented with Mosquitto MQTT[2]. The operators are implemented in Java using the DSP framework Apache Edgent[3]. Each operator is executed as an independent instance of the Apache Edgent framework. The selectivity for operators 1 to 5 are 9.5%, 0.4%, 8.7%, 52.4% and 117.1% respectively, and the transformation on the data stream applied by each operator are described as follows:

- **Language Filter** (Op 1): filters and discards every tweet that is not in English.
- **Special Characters Filter** (Op 2): removes non-letter characters from the tweet text.
- **Non-sentiment Words Filter** (Op 3): removes irrelevant or non-sentiment words (*e.g.* the, and, or).
- **Positive/Negative Words Counter** (Op 4): creates a score and counts the number of words with positive and negative sentiments.
- **Scorer** (Op 5): scores the sentiment of the tweet based on the number of positive and negative words.

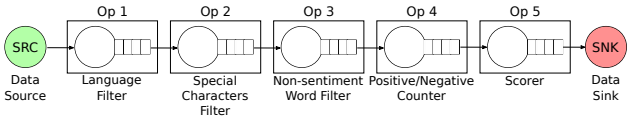

Fig. 6. The operator graph for the sentiment analysis application.

C. Scenarios

We designed six scenarios to compare the estimated and the actual throughput (Table II), each with a different deployment plan for the sentiment analysis application (Fig. 6) considering the infrastructure (Fig. 5). The deployment proposal for each scenario differs regarding the operator assigned for the sequence of each layer. *Data source* and *data sink* have fixed position at the *IoT* and *Cloud* sequence respectively. Also, each sequence has at least one operator; therefore Op 1 is always in the *IoT* sequence, and Op 5 is always in the *Cloud* sequence. The sequence of each layer is replicated for all devices of the layer.

The processing requirements of the operators change in each scenario. The requirements are defined after profiling the application execution according to the deployment plan for the scenario, where the processing capacities of the resources are shared among the operators deployed on the device. Since Op 1 reads the tweets from a data set, the rate at which it reads tweets is affected by the processing requirements of the operator, hence the generation rates change according to the scenario.

The evaluation executed the DSP application for 1440 seconds for each scenario. This time is large enough for the application to execute beyond the warm-up phase, a period needed for the application to achieve a state where all the operators are processing messages at a given time. From observation, the noise created by the warm up phase always lasts less than 300 seconds. Therefore, to ensure that we collect data during the stable phase, we disregard the first 300 seconds.

[1]https://www.openvswitch.org/
[2]https://mosquitto.org/
[3]http://edgent.apache.org/

TABLE II
DEPLOYMENT SCENARIOS.

Scenario	IoT	Edge	Cloud
1	$Op\ 1\ Op\ 2\ Op\ 3$	$Op\ 4$	$Op\ 5$
2	$Op\ 1\ Op\ 2$	$Op\ 3\ Op\ 4$	$Op\ 5$
3	$Op\ 1$	$Op\ 2\ Op\ 3\ Op\ 4$	$Op\ 5$
4	$Op\ 1$	$Op\ 2\ Op\ 3$	$Op\ 4\ Op\ 5$
5	$Op\ 1$	$Op\ 2$	$Op\ 3\ Op\ 4\ Op\ 5$
6	$Op\ 1\ Op\ 2$	$Op\ 3$	$Op\ 4\ Op\ 5$

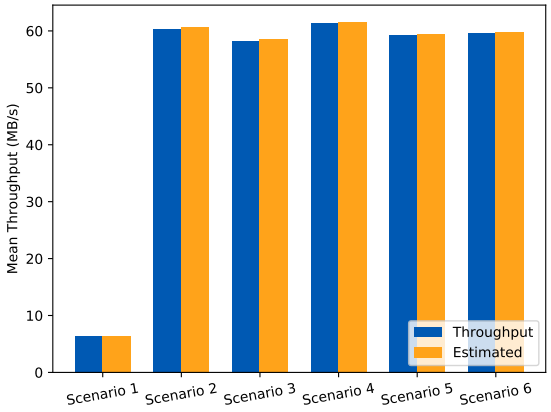

Fig. 7. Mean throughput and estimation.

V. PERFORMANCE EVALUATION RESULTS

The average of actual and estimated throughput is depicted in Fig. 7. Once the system reaches a stable state, the throughput does not vary much. The estimated throughput is close to the actual throughput. Even in the first scenario that produced a very low throughput, the model estimated a close value.

The reason for the low throughput under the first scenario is that the deployment plan deploys three operators in a device with constrained capacity (*i.e.* a Raspberry PI). The third operator in particular, that removes non-sentimental words from tweets, has a significant impact on the application performance. It is very demanding in terms of processing capacity. As it runs on a constrained device, it affects the throughput of the whole application. This reinforces the importance of setting the right requirements for each operator.

Another indicator of the quality of the proposed model is the error of less than 1% in the difference between the estimation and the collected values. We also computed the mean square error to highlight estimated outliers and verify the quality of results. The lower the values the better the quality of the estimation. As depicted in Fig. 8, our estimation achieved low values reinforcing the precision on the proposed model.

VI. RELATED WORK

The Cloud is often the infrastructure of choice for deploying DSP applications. The DICE framework [16] is a tool to help

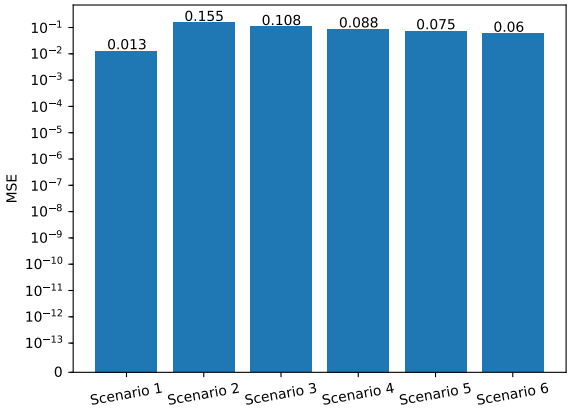

Fig. 8. Mean square error.

in the development of analytical applications to Cloud deployment, including DSP for the Storm framework. The framework also provides a module for estimating application performance based on a deployment plan, but without estimating requirements to improve the performance and disregarding network constraints. Sun and Rui [12] estimate the parallelism level of operators to handle a given event arrival rate whilst respecting the end-to-end latency. They deal with load variations and reschedule operators in the critical path. Instead of estimating the parallelism level of a operators, Pacaci and Özsu [13] propose to balance the load between different instances of a given operator. Li *et al.* [14] propose a framework to predict the end-to-end latency of DSP applications. The prediction is given by a Support Vector Regression model based on collected statistics from a round-robin deployment. The prediction determines the parallelism level and the position of each operator, but not the bandwidth and processing requirements. Souza *et al.* [15] propose an architecture for deploying DSP applications where the operators are adapted to create event batches to increase the application throughput. The batch size along with the deployment are constant adapted based on the load and execution metrics.

Existing work also focuses on providing elasticity on the Cloud. After profiling the execution of a DSP application, De Matteis and Mencagli [17] determine the proper parallelism level for each operator in order to handle load variations. Floratou *et al.* [18] introduce an architecture for automatically scaling-in and out the requirements of the DSP application in such a way that maximizes the throughput without over-provisioning, while respecting the requirements of a user-defined policy. Runsewe and Samaan [19] evaluate the historical performance of an application and, based on a layered multi-dimensional hidden Markov model, determine the required resources to scale up or down the application based on the arrival rate load. Sun *et al.* [6] presented a similar solution, where after profiling the execution of the application, the DSP graph is optimized in terms of processing requirements and

parallelism level of operators.

With respect to Fog computing, Sajjad *et al.* [20] propose a deployment solution to offload operators from the Cloud to minimize the end-to-end latency and costs of using this infrastructure without accounting for processing requirements. Only the parallelism level of each operator is provided by the user. Cardellini *et al.* [5] present a deployment solution for Fog computing that estimates the best parallelism level of each operator, but requires user provision of processing requirements for each operator. Even though it is not DSP oriented, another work [21] proposes a framework for deploying IoT applications on the Fog, but the generality of the proposed model does not account for characteristics of DSP applications, leading to inefficient deployments. The proposed model accounts for estimates of the throughput of DSP applications deployed only on the Cloud or exploring the location-aware from Fog, considering characteristics of the application and the infrastructure. This model could be applied to many of the presented deployment solutions to estimate parallelism level, processing and bandwidth requirements to maximize the application throughput.

VII. Conclusion

The deployment quality of DSP applications is measured either by throughput or end-to-end latency. These are metrics that the deployment solutions aim to improve, and this improvement is limited by the quality of the model employed to the metric. When considering a distributed environment such as Fog computing, building these models becomes challenging. In this work, we presented a throughput model that accounts for characteristics of Fog computing, the DSP application and how they interact to produce the throughput.

Results of experiments realized in a real environment show that the proposed model estimates the throughput with less than 1% error. This result is an indication that the proposed model considers most of the variables affecting the throughout, and if applied to deployment solutions will result in deployments with high throughout. Even further, this precision opens the possibility for this model to be used by deployment solutions to maximize the throughput, by estimating the exact parallelism level, processing and bandwidth requirements for the operators. These are requirements provided by users with no or little expertise. The next step on this research is to use this model as the cornerstone for a deployment solution that estimates the requirements to maximize the throughput based on characteristics of the DSP application.

Acknowledgements

This work is partially supported by “Fonds Recherche” via the emerging project entitled “Algorithmes pour le placement et la reconfiguration des services de traitement de flux de données pour des applications IoT”.

References

[1] 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.

[2] L. Xu, B. Peng, and I. Gupta, “Stela: Enabling stream processing systems to scale-in and scale-out on-demand,” in *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 22–31.

[3] E. G. Renart, J. Diaz-Montes, and M. Parashar, “Data-driven stream processing at the edge,” in *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*. IEEE, 2017, pp. 31–40.

[4] D. Puthal, M. S. Obaidat, P. Nanda, M. Prasad, S. P. Mohanty, and A. Y. Zomaya, “Secure and sustainable load balancing of edge data centers in fog computing,” *IEEE Communications Magazine*, vol. 56, no. 5, pp. 60–65, 2018.

[5] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, “Joint operator replication and placement optimization for distributed streaming applications,” in *proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2017, pp. 263–270.

[6] D. Sun, H. Yan, S. Gao, X. Liu, and R. Buyya, “Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams,” *The Journal of Supercomputing*, vol. 74, no. 2, pp. 615–636, 2018.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 15–26.

[8] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[9] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, “Optimal operator placement for distributed stream processing applications,” in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 69–80.

[10] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, “Quantifying the impact of edge computing on mobile applications,” in *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*. ACM, 2016, p. 5.

[11] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, “Timestream: Reliable stream computation in the cloud,” in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 1–14.

[12] D. Sun and R. Huang, “A stable online scheduling strategy for real-time stream computing over fluctuating big data streams,” *IEEE Access*, vol. 4, pp. 8593–8607, 2016.

[13] A. Pacaci and M. T. Özsu, “Distribution-aware stream partitioning for distributed stream processing systems,” in *Proceedings of the 5th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. ACM, 2018, p. 6.

[14] T. Li, J. Tang, and J. Xu, “Performance modeling and predictive scheduling for distributed stream data processing,” *IEEE Transactions on Big Data*, vol. 2, no. 4, pp. 353–364, 2016.

[15] P. R. de Souza, K. J. Matteussi, J. C. dos Anjos, J. D. dos Santos, C. F. R. Geyer, and A. da Silva Veith, “Aten: A dispatcher for big data applications in heterogeneous systems,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 585–592.

[16] “Dice consortium tech. reports,” 2016-2017. [Online]. Available: http://www.dice-h2020.eu/deliverables/

[17] T. De Matteis and G. Mencagli, “Proactive elasticity and energy awareness in data stream processing,” *Journal of Systems and Software*, vol. 127, pp. 302–319, 2017.

[18] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, “Dhalion: self-regulating stream processing in heron,” *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1825–1836, 2017.

[19] O. Runsewe and N. Samaan, “Cloud resource scaling for big data streaming applications using a layered multi-dimensional hidden markov model,” in *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*. IEEE, 2017, pp. 848–857.

[20] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, “Spanedge: Towards unifying stream processing over central and near-the-edge data centers,” in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 168–178.

[21] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos, “Fog based framework for iot service provisioning,” in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2019, pp. 1–6.