A Throughput Model for Data Stream Processing on Fog Computing
Felipe Rodrigo de Souza, Marcos Dias de Assuncao, Eddy Caron

To cite this version:

HAL Id: hal-02140851
https://hal.archives-ouvertes.fr/hal-02140851
Submitted on 27 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 itself – 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.
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 $I = (\langle D \cup H \rangle, C)$, 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 $A = (O, E)$ 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
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 \((D_j, S_j)\), 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 \(\text{Sink}(l)\) and \(\text{Source}(l)\). The function \(\text{Source}(l)\) provides the set of operators that receive data from operators in other layers, or from data sources in the IoT layer. \(\text{Sink}(l)\) 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 \(\ell(j)\).

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 $i \in (D \cup H)$, 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 $\lambda_i^j$ (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 $P(j',j)$ that $j'$ will send the outgoing stream to $j$.

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

$$\lambda_i^j = \sum_{j' \in Prv(j)} \sum_{k \in M(j')} S_{i,k}^{j,j'}$$  \hspace{1cm} (2)

$$S_{i,k}^{j,j'} = \begin{cases} P(j',j) \times DR_i^j, & \text{if } i = k \\ \min \left[ \frac{P(j',j) \times DR_i^j}{s_{PP}(P(T(j'),j),k)}, Net(k,i) \right], & \text{otherwise} \end{cases}$$  \hspace{1cm} (3)

The departure rate $DR_i^j$ (Eq. 4) applies the selectivity $S_i$ 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.

$$DR_i^j = AR_i^j \times (1 - S_i)$$  \hspace{1cm} (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 $PP(j)$.

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 $Op 4 \rightarrow Op 6 \rightarrow Op 7$. While $Op 6$ starts to process $M 1$, $Op 4$ starts to process $M 2$. After 1 time unit $Op 4$ finishes $M 2$ and stores it in a queue while $Op 6$ finishes processing $M 1$. 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.

$$PT(j) = \frac{AR_{first}(PP(j))}{D_{first}(PP(j))} + Q(j) = \sum_{k \in PP(j)} \frac{AR_k}{D_k}$$  \hspace{1cm} (5)

where $Q(j)$ is given by:

$$Q(j) = \sum_{k \in PP(j)} \left\{ \begin{array}{ll} \frac{AR_{i+k}}{D_{ik}}, & k = first(PP(j)) \\ \max \left[ \frac{AR_{i+k}}{D_{ik}}, \frac{AR_{i+k+1}}{D_{ik+1}} \right], & \text{otherwise} \end{array} \right.\hspace{1cm} (6)$$

The proposed model computes the throughput for each layer $l$ (Eq. 7), based on the departure rate of all operators $j \in$ Sink($l$) 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).

$$Rate(l) = \sum_{j \in \text{Sink}(l)} \frac{DR_{l,j}}{\max(PT(j), 1)}$$  \hspace{1cm} (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.
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 0.4 ms [10]. The gateway is a server with an Intel® Xeon® ES-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 $\approx 2.3$Gb/s. The latency between the gateway and the edge server is configured as $\approx 24$ms and the latency between the edge server and the Cloud is $\approx 50$ms [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.

**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/
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 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 Mencaglì [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 throughput, and if applied to deployment solutions will result in deployments with high throughput. 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