Investigating Edge vs. Cloud Computing Trade-offs for Stream Processing

Pedro Silva, Alexandru Costan, Gabriel Antoniu
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
pedro.silva@irisa.fr, alexandru.costan@irisa.fr, gabriel.antoniu@inria.fr

***Abstract*—The recent spectacular rise of the Internet of Things and the associated augmentation of the data deluge motivated the emergence of Edge computing as a means to distribute processing from centralized Clouds towards decentralized processing units close to the data sources. This led to new challenges in ways to distribute processing across Cloud-based, Edge-based or hybrid Cloud/Edge-based infrastructures. In particular, a major question is: how much can one improve (or degrade) the performance of an application by performing computation closer to the data sources rather than in the Cloud? This paper proposes a methodology to understand such performance trade-offs and illustrates it through experimental evaluation with two real-life stream processing use-cases executed on fully-Cloud and hybrid Cloud-Edge testbeds using state-of-the-art processing engines for each environment. We derive a set of take-aways for the community, highlighting the limitations of each environment, the scenarios that could benefit from hybrid Edge-Cloud deployments, what relevant parameters impact performance and how.**

***Index Terms*—Edge Computing, Edge Analytics, Fog Computing, Cloud Computing, Stream Processing, Flink, Edgent.**

I. Introduction

In the last two decades, the Cloud became the preferred processing infrastructure for many applications thanks to features like elasticity, scalability and on-demand cost models. However, it has been challenging to use Cloud computing on some new types of real-time distributed applications which are capable of generating huge amounts of data and which demand low-latency processing responses. For instance, in the context of large Internet of Things (IoT) applications, the connections between the many sensors and the Cloud resources may be an issue because of the potential high-latency of the Internet. Similarly, it may be impossible to send the data to the Cloud due to data national security laws or simply because an Internet connection is not available. Furthermore, data transmission costs (e.g., Cloud provider fees, carrier costs) could make a business solution impractical.

Edge computing is a new paradigm which aims to address some of these issues. The key idea is to leverage computing and storage resources at the "edge" of the network, i.e., on processing units located close to the data sources [1]. This allows applications to outsource data processing from the main (Cloud) processing data centers to the Edge. For instance, a home assistant may perform a first lexical analysis before requesting a translation to the Cloud. However, Edge computing is not a silver bullet: issues like node volatility, limited processing power, high latency between nodes, fault tolerance and data degradation may impact applications depending on the characteristics of the infrastructure.

This work is supported by the ANR OverFlow project (ANR-15-CE25-0003). All experiments were carried out using the Grid'5000 testbed.

In this paper we are interested in understanding the conditions that enable the usage of Edge or Cloud computing in order to reduce the time to results and the associated costs. While some state-of-the-art approaches advocate either "100% Cloud" or "100% Edge" solutions, the relative efficiency of a method over the other may vary. Intuitively, it depends on many parameters, including network technology, hardware characteristics, volume of data or computing power, processing framework configuration and application requirements.

Some solutions advocate hybrid Edge-Cloud processing [2], however, in order to split the data processing across those two infrastructures it is necessary to understand the consequences of their interaction. This paper makes a first step in this direction with the following contributions:

- a methodology to understand the performance of Edge and Cloud applications;
- a thorough experimental evaluation with two representative real-life stream processing use-cases;
- a set of take-aways for the community, highlighting the limitations of each environment, the scenarios that could benefit from hybrid Edge-Cloud deployments and how various parameters of interest impact performance.

The rest of this paper is organized as follows. We first contextualize this work (Section II), then we present the applications, datasets and workloads used in our experiments (Section III). We detail our experimental methodology (Section IV) and setup (Section V), followed by the evaluation of the experiments (Section VI and VII). We formulate our final conclusions in Section VIII.

II. Background and Related Work

A *data stream*, or simply a *stream*, is an unbounded and continuous flow of data. Examples include data produced by sensors or user events from social networks such as tweets, postings and comments. Stream processing applications typically collect data from the Edge of the networks, where they are produced and eventually pre-processed. Clouds are used as centralized, main processing data centers, where the data are shipped for further, complex analytics (e.g., using batch processing). In this section, we introduce the main concepts related to stream processing in Edge and Cloud environments.

A. Stream processing at and near the Edge

Edge computing is the collection of technologies that allows computation to be performed at the Edge of a network [3], i.e., on the set of resources *between* the data sources and the Cloud data centers. Edge computing can be applied to a large variety of application domains, and, in particular, in fields related to the IoT. For instance, events emitted by sensors installed on trucks are processed locally (i.e., in the trucks) in order to reduce latency and transmission costs [4]. Similarly, anomalies on water transmission pipes are detected locally in order to reduce response latency [5]; the performance of image and video processing on mobile devices is improved by offloading part of the computation to Edge servers [6].

Edge computing is enabled by several platforms that target software deployment, data processing and system monitoring pushed by the open-source community [7]–[10] and by large technology companies [11], [12]. In this work, we use Apache Edgent [9], a light-weight open source Java library that can be easily deployed on Edge devices and nodes.

Fog computing, similarly to Edge computing, is a paradigm that enables computation to be performed near the data sources. The essential difference between Edge and Fog computing is the exact location where computation is performed. In Edge computing, this usually occurs directly on Edge *devices*, like sensors and actuators. In Fog computing, it is commonly performed on some dedicated *nodes*, typically located between the Edge and the Cloud such as in the local area network connecting Edge devices or on the network gateways (i.e., at the left of the dotted line in Figure 1). Although there is no consensus about those two concepts and many authors use them interchangeably, we make this distinction for the sake of clarity. Our evaluation framework can be deployed on both *Edge devices* and *Edge nodes* (the latter corresponding to what could be referred as Fog resources), as detailed in Section IV.

B. Stream computing in the Cloud

Once data are collected from the Edge, the typical stream computing pipeline consists of two stages executed in the Cloud and depicted in Figure 1: *ingestion* and *processing*.

Data ingestion serves mainly to aggregate, buffer and log data streams before they are consumed by the processing phase. Several mature open-source systems perform data ingestion, e.g., Pivotal RabbitMQ [13] and Apache Kafka [14]. They rely on publish-subscribe mechanisms to handle streams and to store them a in fault-tolerant durable way. Kafka is the most widely used and is our choice for this paper.

Stream processing executes data analytics on streams. Very often, streams must be processed in real-time, due to, mainly, the requirement of low-latency data processing and analysis of stream applications. A large set of stream processing engines were proposed specifically to address these challenges. Most notable examples are from the Apache ecosystem: Flink [15], Spark [16], and Storm [17]. In this work, we leverage Apache Flink, a reference open source, low-latency and high-throughput stream processing engine offering exactly-once semantics.

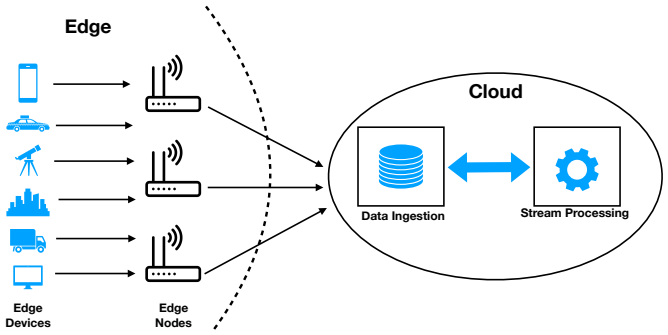

Fig. 1. Typical stream computing pipeline from the Edge to the Cloud involving data ingestion and stream processing [18].

C. Challenges of hybrid Edge and Cloud computing

Edge computing can be a solution for many scenarios, as discussed in Section II-A. However, it may often have a negative impact on application performance or cost.

Let us illustrate the case of improving the performance of windowed aggregations [19], [20] by processing part of the data locally on the Edge before sending it to the Cloud. In this case, there is a trade-off between computation performance (i.e., execution time) and data transmission (i.e., cost). Doing all the aggregation on the Edge results in less data sent to the Cloud, however, this choice can also impact the time needed at the Edge to treat each local aggregation.

The ultimate goal of a *hybrid* Edge and Cloud deployment is to have an online/real-time front-end for processing on the Edge, close to where data are generated, while the Cloud will only be used for offline back-end processing, mainly dealing with archival, fault tolerance and also further processing that is not time-critical. This hybrid approach enables Edge analytics to detect *what* is happening with a monitored object, while Cloud analytics allows to understand *why* this is happening.

We can find work in the state-of-the art aiming at analyzing Cloud-only based data processing scenarios [21], [22] or Edge-only based data processing scenarios [23], [24]. However, to the best of our knowledge, there is no previous work proposing an experimental methodology and analysis for *hybrid* Cloud and Edge processing.

.

III. Datasets and Workloads

We use two different stream processing scenarios for our evaluation. The first one uses data from the New York City Taxi and Limousine Commission (TLC) rides [25]. The second one consists of CCTV footage from the University of California San Diego [26]. The main difference between the two datasets lies in their data items size. Records in the TLC dataset are about 300B, making it representative for scenarios involving sensors that send small messages in Smart Cities. In contrast, video frames from CCTV dataset are about 25KB, matching applications that process larger messages, such as log analysis.

1) TLC Dataset: It is composed of records, emitted at the end of rides, containing information about taxi rides such as driver identification, pick-up and drop-off times, and locations [25]. We use taxi rides data from 2013 and the size of the dataset is around 20GB. The scenario we implemented consists in finding the busiest driver every two hours. To do that, five operations need to be performed: *(1)* spurious records (records with invalid data) are *filtered*; *(2)* a percentage FR of records, defined as a parameter of interest, is *filtered*; *(3)* pickup and drop-off positions and times are used to *calculate* the travel distance and average speed; *(4)* a *windowed* aggregation gets the sum of working times for each driver in the last two hours; *(5)* a *windowed* aggregation calculates the driver who drove the most during the last two hours. In Edge scenarios, operations (1), (2) and (3) are performed on the Edge.

2) CCTV Dataset: It is composed of footage recorded within the university campus by CCTV cameras capable of recording 10 frames per second. The original dataset has around 700MB. In the implemented scenario, we identify the most crowded places in the campus every minute. For doing that, we perform three operations: *(1)* the number of persons in each frame is *counted*; *(2)* a *windowed* aggregation gets the maximum number of persons observed in the last WS seconds for each camera, where WS is a previously defined parameter of interest; *(3)* the maximum number of persons, for all cameras, in the last two minutes is calculated using a *windowed* aggregation. For counting the number of persons in a frame, we use techniques of background subtraction [27] and k-nearest neighbors with OpenCV [27]. In Edge scenarios, operations (1) and (2) are performed on the Edge.

IV. Methodology

In order to understand the impact of Edge and Cloud environments we analyze the results of series of experiments. An experiment is a set of executions of one of the considered scenarios (TLC and CCTV). Every scenario has a set of *parameters of interest* used to configure the applications. Our objective is to observe the behavior of some application performance metrics, which we refer to as *metrics of interest*, when combining the different values of the parameters of interest. In this section, we detail these parameters and metrics of interest and present our targeted experiments.

A. Parameters of interest

We select a set of 6 parameters whose values we vary to observe their impact on the perceived application performance. The first 4 parameters are common to both scenarios:

- *type of processing (T)*: it indicates the deployment environment: Cloud or hybrid Edge-Cloud;
- *number of data producers per gateway (P)*: it counts the amount of Edge devices in the infrastructure (and consequently the volume of generated data);
- *number of gateways (GW)*: it defines the number of nodes that will be responsible for gathering data from data sources and for uploading data to the Cloud;
- *bandwidth (BW)*: it measures the bandwidth between Edge gateways and Cloud proxy servers.

The last 2 parameters are specific to each scenario:

- *filtering rate (FR)*: specific to the TLC scenario, it defines the percentage of data filtered out of the stream;
- *window size (WS)*: specific to the CCTV scenario, it indicates for each camera the amount of video frames to aggregate before pre-processing them.

TABLE I
Experiment configuration for the TLC scenario.

Scenario	Parameters	Values
CCTV+TLC	T	$\{Edge, Cloud\}$
CCTV+TLC	P	$\{20, 30, 40\}$
CCTV+TLC	GW	$\{30\}$
CCTV+TLC	BW (Mb)	$\{10, 100, 1000\}$
TLC	FR (%)	$\{30, 50\}$
CCTV	WS (seconds)	$\{5, 10\}$
CCTV+TLC	Timeout (s)	420
	Total experiments	36

B. Metrics of interest

We measure a set of 3 metrics to evaluate the behavior of the considered streaming scenarios: *Edge-to-Cloud latency*, *Edge-to-Cloud throughput*, and *end-to-end throughput*.

1) Edge-to-Cloud latency: describes the time, in seconds, a record takes to travel from the device which produced it to the ingestion system (e.g., Apache Kafka). This metric is directly impacted by the amount of data produced on the Edge and the network bandwidth between Edge gateways and the Cloud. Larger filtering rates or window sizes result in more processing in the Edge and less data being sent to the Cloud, hence, less congestion.

2) Edge-to-Cloud throughput: indicates the amount of records that arrive at the ingestion system per second. This can be measured in records or MB per second. Similarly to the Edge-to-Cloud latency, this metric depends on the amount of produced data, the connectivity between Edge and Cloud, and on the scenario specific parameters.

3) End-to-end throughput: measures the rate at which processed data is output by the Cloud engine (e.g., Apache Flink). This metric is affected by all the parameters of interest.

C. Experiment workflow

There are four steps in each experiment: *(i)* generation of the experiments metadata by combining the parameters of interest values presented in Tables I; *(ii)* copy of experiment datasets to data sources hosts; *(iii)* execution of the experiments and logging of the resource usage of each machine; and *(iv)* execution of the data analyzers responsible for gathering the processed data.

V. Experimental Setup

The infrastructure used to deploy the scenarios and run the experiments is summarized in the schema in Figure 2. Once produced by the data sources, data will be either processed entirely in the Cloud, or partially processed at the Edge and

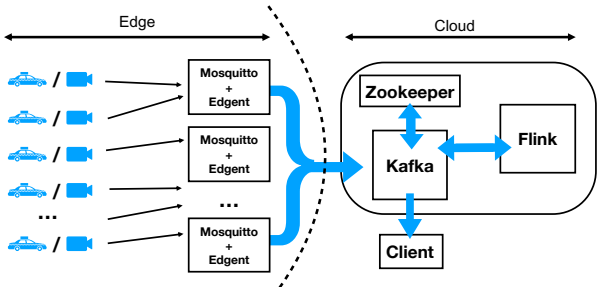

Fig. 2. Simplified model of the infrastructure deployed for the experiments.

then sent to the Cloud. There are several components handling this workflow: data producers, Edgent nodes and gateways compose the Edge infrastructure; Kafka brokers, Zookeeper servers, Flink nodes and clients compose the Cloud infrastructure. We explain each of them in detail in the following.

A. The Edge infrastructure

Data producers are used to simulate the generation of data by sensors/actuators (e.g., taxis and CCTV cameras). In both scenarios (cf. Section III), sensors are Java programs, implemented with the Eclipse Paho library [28]. They read data from files and forward it to the gateways, using the Message Queuing Telemetry Transport (MQTT) protocol [29], the most used IoT protocol in the literature.

Gateways are responsible for receiving data from a set of sensors and transferring it to a main data center in the Cloud. They are implemented as Eclipse Mosquitto servers [30], equipped with a set of Kafka producers for data forwarding. In our experiments, they are responsible for processing the data and sending the results to the Cloud using Apache Edgent.

The set of data producers and gateways compose the Edge infrastructure. The amount of data collected varies depending on the experiment.

B. The Cloud infrastructure

Apache Kafka [14] is used to ingest streams coming from the sensors before being processed. In summary, Kafka stores arriving messages into partitions which, in their turn, are organized into topics. The clients that write and read data from partitions are called producers and consumers, respectively.

Apache Zookeeper [31] handles Kafka's configurations and metadata.

Apache Flink [15] is a high-throughput, low-latency stream processing engine for the Cloud. In our scenarios, it is responsible for getting data from Kafka, processing it and storing the results back to Kafka.

The Client is written in Java and it is responsible for downloading processed data from Kafka and analyzing it in order to calculate the metrics of interest.

VI. Evaluation

In this section, we discuss key points about the implementation and execution of the experiments and interpret the results.

A. Infrastructure and frameworks specifications

For our experiments we use 30 nodes from the *paravance* and the *parasilo* clusters of the Grid'5000 testbed. The nodes are equipped with Intel Xeon processors (Haswell, 2.40GHz, 2 CPUs/node, 8 cores/CPU) and 128GB of RAM. For managing the network bandwidth and latencies we use virtual LANs and the Linux traffic control (*tc*) tools. Every node has its resources (CPU, memory, disk, I/O and network) monitored and logged using the Linux *dstat* command.

We deploy Apache Flink v1.6 on 4 machines per cluster except when indicated otherwise. We use Apache Kafka v1.0 with 4 brokers connected to one Zookeeper server. Kafka was configured with 2 topics with 32 partitions each, one for ingesting data sent by the gateways and another one to receive data produced by Flink jobs. In addition, we set *batch.size* and *linger.ms* with 200KB and 50ms, respectively. Those values were defined through experimentation and are used throughout all experiments.

B. Data preparation

The experimental workflow for both datasets is essentially the same. The only differences are in the way data are generated and the data transmission frequencies of each source.

In the CCTV scenario there are, originally, around 700MB of data. To be able to run our larger scale experiments without having to generate new video frames, before execution, each data source picks randomly around 6,000 frames from the original dataset. In that way, even if the resulting videos do not make sense for human eyes, they are distinct and valid for data processing. Once the data is available in the data sources, video frames are read every 100ms, wrapped with their creation time stamp, and sent to the Mosquitto server.

In the original TLC dataset there are about 20GB of taxi ride logs, which is enough for running our experiments without having to generate data. TLC records come already with a creation time stamp (i.e., the time the taxi ride ended), hence, there is no need to redefine it. In order to compress an entire year of data into our experiments, we mapped each minute in the original dataset to 5ms.

When an experiment finishes its execution, two Kafka consumers deployed on dedicated nodes get a sample of its input topic and the entire output topic. Only a sample of the input topic is recovered because in Cloud only scenarios, this may be very large and would demand too much time and resources to read it entirely.

C. Analysis

In this section we discuss how the parameters of interest impact the performance of the TLC and CCTV workloads and highlight some key differences between the two.

Impact of the bandwidth on the Edge-to-Cloud throughput. In Figure 3, we plot the Edge-to-Cloud throughput of the CCTV workload when all the stream processing is performed on the Cloud. When the bandwidth (BW) is 100Mb/s or 1000Mb/s, there is a correlation between the throughput and the number of data sources (P), in the X axis. Note, however,

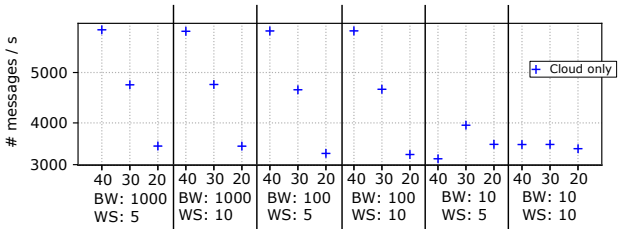

Fig. 3. Edge-to-Cloud throughput for different configurations of Cloud only CCTV experiments. On the X axis we vary the number of data sources (P).

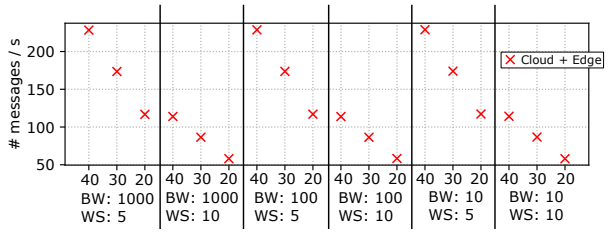

Fig. 4. Edge to Cloud throughput (messages/s) for different configurations of Cloud + Edge CCTV experiments. *P* varies on the X axis.

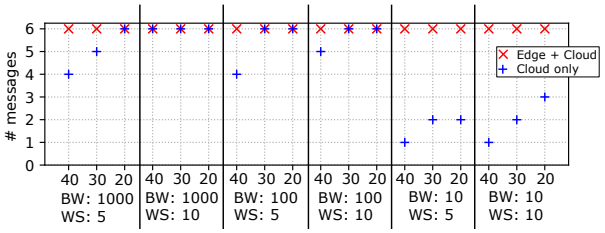

Fig. 5. Output of experiments in terms of number of messages for CCTV experiments. Note that the maximum number of output messages for this experiment is 6.

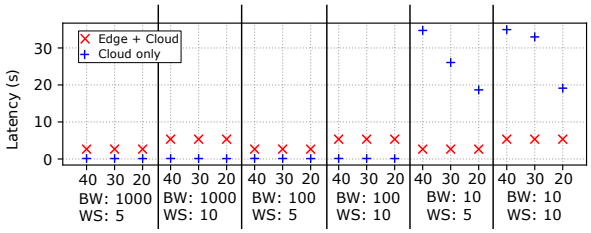

Fig. 6. Latency of experiments for CCTV experiments.

that the correlation disappears when BW is 10Mb/s and the throughput keeps close to constant. This behavior of data invariance followed by an augmentation of the data volume characterizes a *bandwidth bottleneck*.

In Figure 4, we illustrate the throughput between Edge and Cloud for CCTV when the stream processing is performed partially at the Edge and on the Cloud (cf. Section III). There is a correlation between throughput, window size (WS) and number of data sources (P) in which more data and smaller windows result in increased throughput. In this scenario, the bandwidth does not affect the throughput and this can be verified by checking that all points with the same P, W and BW have similar throughput. In this case the bandwidth is no longer a bottleneck because data is aggregated in the Edge before being transmitted. Hence, only a fraction of the total amount of data, which in this case varies between 1% and 2%, is actually sent to the Cloud.

Impact of the bandwidth and of the number of data sources on the end-to-end throughput. In Figure 5 we illustrate the number of messages produced by Flink, which corresponds to the amount of solutions calculated by each experiment in Cloud only and hybrid Cloud and Edge scenarios.

Analyzing exclusively the Cloud-only throughput in Figure 5, we can observe two interesting trends. Firstly, when the value of BW is 10Mb/s, the throughput is reduced and it follows an almost constant pattern. Again, this happens because of a bandwidth bottleneck: in our experiments, data is processed by Flink at *event time*, i.e., the time observed in each message dictates the time flow. In this case, it means that the processing clock is behind the wall clock: the data stream arrives slowly in the Cloud and, consequently, "time" passes slowly. Secondly, when BW is 100Mb/s or 1000Mb/s, there are experiments where P is 30 or 40 and the throughput also reduces. However, in this case the bottleneck is not the bandwidth, but the number of consumers. As the input throughput is too high, a larger Flink cluster is needed. This issue disappears when we increase its size, as the application manages to process the workload in time.

Impact of the bandwidth and of the number of data sources on the Edge-to-Cloud latency. The bandwidth bottleneck can also be identified when analyzing the median of the application latency, illustrated in Figure 6. In this Figure, we identify (i) the overhead of data aggregation as the distance between Edge + Cloud and Cloud only latencies, and (ii) the bandwidth bottleneck issues – when bandwidth is 10Mb/s – which impacts the latency of Cloud only experiments.

Impact of the record size on the Edge-to-Cloud throughput. We plot the Edge-to-Cloud throughput of the TLC workloads grouped by BW, FR and P in Figure 7. We identify similar patterns to those observed in Figures 5, however the effects of bandwidth bottlenecks are smaller. This is due to the different message sizes of each scenario. While video frames from the CCTV dataset are around 25KB, TLC records are about 300B. Hence, even if there are more messages flowing from Edge to the Cloud, the influence of the bandwidth size on the throughput is smaller.

An important detail that links the performance of hybrid Edge and Cloud experiments in Figures 4 and 7 is that WS and FR affects the amount of data uploaded to the Cloud in an almost linear fashion.

VII. Discussion

Edge and Cloud computing can and do work well together. However, Edge computing is more suited for purpose-built systems with special needs, while Cloud computing remains more general. In this section, we present our main take-aways for the community about the interaction between the two.

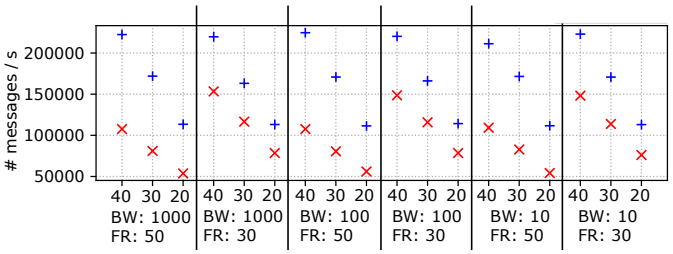

Fig. 7. Throughput Edge-to-Cloud for TLC workloads for different parameter configurations.

Bandwidth is a key parameter for hybrid Edge-Cloud processing. As expected, we observe that, among all parameters of interest, *the bandwidth impacts every metric*. However, its influence can be offset by *controlling the workload size and the Edge gateways parallelism*.

Complex, stateful operations are not suitable for Edge only. Processing all data in the Edge, in complex scenarios is usually not possible mainly because *Edge nodes and devices have only a partial view of the entire data*. Additionally, often Edge machines have restricted resources and battery life, and are more susceptible to errors and virtual or physical attacks than Cloud servers.

Hybrid Edge-Cloud processing speeds-up stateless and simple stateful operations. As observed throughout the previous section, when it comes to stateless or simple stateful operations, pre-processing data on the Edge before sending them to the Cloud may have some very interesting benefits. In our experiments, we observe that *Edge computing helps reduce the amount of data being uploaded to the Cloud* and, consequently, improves the overall performance, in terms of end-to-end throughput.

Edge processing reduces network and Cloud resources costs. While in many cases the Edge processing cannot bring performance benefits to the overall application, *it can reduce its incurred costs*. By filtering and aggregating data locally, the transmissions costs and also the Cloud computing resources costs are reduced (i.e., by sending and processing smaller volumes of data, less processing power in the Cloud is needed).

VIII. Conclusion

The anticipated deployment of 5G networks in the near future will allow to transmit very large amounts of data short distances, with lower latencies and higher speeds. Consequently, Edge processing will become ubiquitous. In this context, identifying and understanding the impact of various parameter settings on the performance of applications executed on hybrid Edge-Cloud environments could trigger a new wave of research aiming to achieve uniform and automated scheduling of tasks on both infrastructures. Besides eliminating such burden from users, this can enable optimized computation placement with respect to network and resource costs.
monetary costs, and examine in this context the extent to

By necessity, our preliminary study only focused on a series of parameters. As future work, we plan to extend and refine our evaluation to take into account energy consumption and which the benefits of using Edge-based or hybrid Edge-Cloud processing can offset the associated costs.

References

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, 2017.

[2] L. Prosperi, A. Costan, P. Silva, and G. Antoniu, "Planner: Cost-efficient Execution Plans Placement for Uniform Stream Analytics on Edge and Cloud," in *WORKS 2018*, 2018.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, 2016.

[4] R. Young, S. Fallon, and P. Jacob, "An architecture for intelligent data processing on iot edge devices," *UKSim-AMSS*, 2017.

[5] S. Kartakis, W. Yu, R. Akhavan, and J. A. McCann, "Adaptive edge analytics for distributed networked control of water systems," in *IEEE IoTDI*, 2016.

[6] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proceeding of MobiSys '13*, 2013.

[7] Y. Qiu, Y. Chen, L. Jiao, and S. Huang, "Rta: Real time actionable events detection as a service," in *IEEE CWS*, 2016.

[8] 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 *IEEE/ACM SEC*, 2016.

[9] Apache, "Apache edgent." [Online]. Available: https://edgent.incubator.apache.org

[10] ——, "Apache edgent." [Online]. Available: https://nifi.apache.org

[11] Amazon, "Aws iot greengrass." [Online]. Available: https://aws.amazon.com/greengrass/

[12] IBM, "Ibm watson iot." [Online]. Available: https://www.ibm.com/internet-of-things

[13] Pivota, "Rabbit mq." [Online]. Available: https://www.rabbitmq.com

[14] Apache, "Apache kafka." [Online]. Available: https://kafka.apache.org

[15] ——, "Apache flink." [Online]. Available: https://flink.apache.org

[16] ——, "Apache spark." [Online]. Available: https://spark.apache.org

[17] ——, "Apache storm." [Online]. Available: http://storm.apache.org

[18] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *IEEE SmartCloud*, 2016.

[19] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing grouped aggregation in geo-distributed streaming analytics," in *HPDC '15*. ACM, 2015.

[20] Y. Zhu Dennis Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," 09 2002.

[21] O. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernández, "Spark versus flink: Understanding performance in big data analytics frameworks," in *IEEE CLUSTER*, 2016.

[22] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *IEEE IPDPSW*, 2016.

[23] A. Das, S. Patterson, and M. P. Wittie, "Edgebench: Benchmarking edge computing platforms," *CoRR*, 2018.

[24] A. Medvedev, A. Hassani, A. Zaslavsky, P. P. Jayaraman, M. Indrawan-Santiago, P. Delir Haghighi, and S. Ling, "Data ingestion and storage performance of iot platforms: Study of openiot," in *Interoperability and Open-Source Solutions for the Internet of Things*, I. Podnar Žarko, A. Broering, S. Soursos, and M. Serrano, Eds., 2017.

[25] B. Donovan and D. B. Work, "Using coarse gps data to quantify city-scale transportation system resilience to extreme events." in *Transportation Research Board 94th Annual Meeting*, 2015.

[26] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[27] OpenCV, "Opencv." [Online]. Available: https://opencv.org

[28] Eclipse, "Eclipse paho." [Online]. Available: https://www.eclipse.org/paho/

[29] Oasis, "Message queuing telemetry transport." [Online]. Available: https://mosquitto.org/

[30] Eclipse, "Eclipse mosquitto." [Online]. Available: https://mosquitto.org/

[31] Apache, "Apache zookeeper." [Online]. Available: https://zookeeper.apache.org/