Compas'2017 : Parallélisme / Architecture / Système
Sophia Antipolis, France, du 27 au 30 juin 2017

Assessing the Impact of Network Bandwidth and Operator Placement on Data Stream Processing for Edge Computing Environments

Alexandre Veith *, Marcos Dias de Assunção, Laurent Lefèvre

Inria Avalon, LIP, ENS Lyon, University of Lyon
46 allee d'Italie
69364 Lyon - France
alexandre.veith@ens-lyon.fr

Résumé
A substantial part of the "big data" generated today is received in near real time and must be promptly processed. Cloud-based architectures for data stream processing comprise multiple software modules or frameworks for data collection, message queueing, and stream processing itself. This modular approach allows each component to grow independently from one another and accommodate changes, but it may increase the end-to-end latency when data events are processed in the cloud. Recent solutions intend to explore the edges of the Internet (*i.e.* edge computing) to perform certain data processing tasks and hence better utilise network resources. This work evaluates the impact regarding network bandwidth while employing frameworks that are commonly used to build cloud and edge-based stream processing solutions.

Mots-clés : Traitement de flux de données, méta-données, analyse de performance, systèmes distribués, informatique en nuages

---

1. Introduction

Today's instruments and services are producing ever-increasing amounts of data that require processing and analysis in order to provide insights or assist in decision making. This data deluge, often called *big data*, poses challenges to existing infrastructure regarding data transfer, storage, and processing. Although application models such as MapReduce have been very popular for batch processing, much of the data generated today is received in near real-time and requires quick analysis. In Internet of Things (IoT) [5, 12], for instance, continuous data streams produced by multiple sources must be handled under very short delays.
The interest in processing data events as they arrive (*i.e.* online) has led to the emergence of several Distributed Stream Processing Engines (DSPEs) such as Apache Storm, Spark, Flink and S4. Under many frameworks, a stream processing application is often a Directed Acyclic Graph (DAG) whose vertices are operators that execute a function over the incoming data and edges define how data flows between them.

*. This work was performed within the framework of the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Clouds are often the target infrastructure for deploying such engines due to their scalability, pay-as-you-go business model and resource elasticity. DSPEs are generally part of a larger architecture that comprises multiple tiers of data collection and processing interconnected by message brokers and queuing systems such as Apache ActiveMQ [1] and RabbitMQ [3], and publish-subscribe solutions including Apache Kafka [2]. This modular design enables tiers to grow at different paces and accommodate changes, but can increase the end-to-end latency and communication cost when treating data events.

More modern solutions intend to exploit the edges of the Internet (*i.e.* edge computing) for performing certain data processing tasks and hence: reduce the end-to-end latency and communication costs, enable services to react to events locally, or offload processing from the cloud [7]. However, the architectures and software frameworks for these environments comprising clouds and micro-data centres located at the Internet edges are still evolving.

In this work, we evaluate the impact and opportunities for deploying stream processing applications using cloud and edge computing. Our study considers DAGs that span both cloud and edge infrastructure. We are interested in demonstrating the benefits of splitting a stream processing graph and spreading operators across available resources.

2. Proposal Architecture

This work considers multiple geographically distributed infrastructures for stream processing in big data environments, as depicted in Figure 1. A data stream flow usually has a source (*i.e.* a resource that creates or collects the data), operators that perform transformations on the data (*e.g.* filtering, mapping and aggregation) and a sink (*i.e.* the destination of the data). In a traditional deployment, all the operators of a data streaming application are placed in the cloud to benefit from virtually unlimited resources. Initially designed to minimise the latency of content delivered to users of mobile devices, edge computing has become an attractive solution for performing certain stream processing operations. By leveraging idle edge resources (*i.e.* sensors and gateways) computations can be performed closer to where the data is generated.

Our effort relates to improving resource utilisation and data movement, which in stream processing can be achieved by deploying data stream operators along the physical path (*source* and *sink*). Instead of hosting all the data stream operators required by a stream processing solution in the cloud itself, we consider a more decentralised approach where the application is decoupled and hosted at multiple geographical locations. However, in this kind of environment, there are several variables (*e.g.* computational power, network bandwidth, network latency, application constraints, network topology) which have a strong impact on the placement decisions and bring complexity to the problem [11].

To optimise the aforementioned environment and deal with stringent stream processing requirements (*i.e.*, events often being handled in the order of seconds or milliseconds), we present an architecture (Figure 2) to optimise the placement of a DAG's elements. In our model, a set of a computational resources will form a group considering resource proximity, which can translate to the number of hops and available bandwidth between the hosts. The resource roles are described as follows:

— **Orchestrator node**: Performs *global decisions*, and stores information about the tasks (*i.e.*, DAG topology) and available containers (*i.e.* environment prepared to receive and process tasks). The global decisions involve task and container placement, as well as system monitoring. The placement considers information about the tasks, the network and the resources.

— **Master node**: Makes *local decisions* (decisions in the group) and group monitoring. The

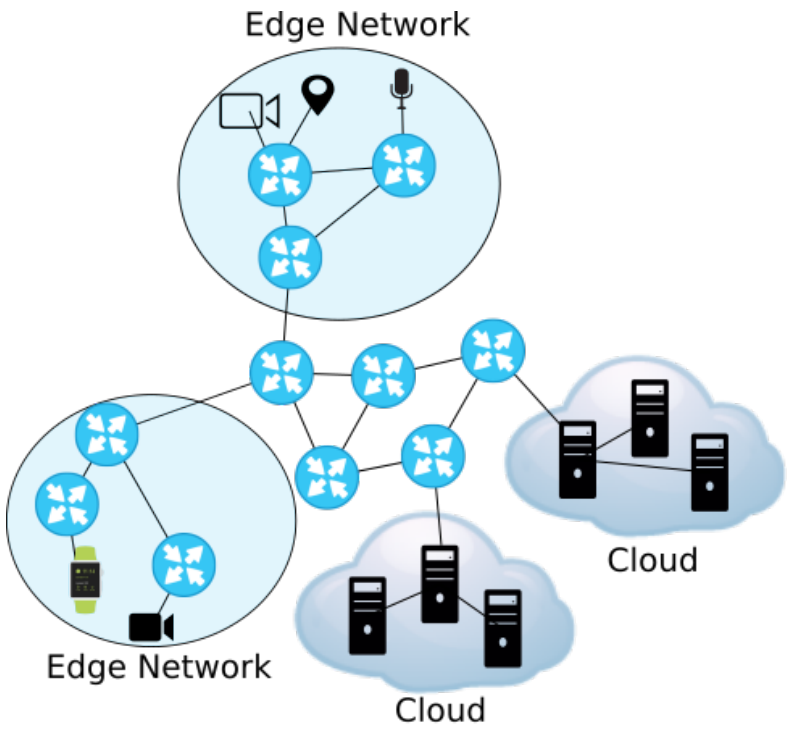


FIGURE 1 – Physical infrastructure which represents the flow between data sources and sinks.

local decisions refer to small adjustments on the local task placement.

— **Worker node**: Hosts the operators and executes functions over the data streams, and/or stores data after processing.

This work focuses on the communication between worker nodes. The flow between the data streams will be performed through the queue's consumption. In this way, a worker will get the data from another worker queue or a source queue. The consumption directions will be given by the Orchestrator or by the Master node. By distributing data stream operators, we would expect to minimise the latency, minimise the amount of data transferred over the wire (*e.g.* fewer headers, serialisation, etc), and reduce the impact of the external network environment (*i.e.* Internet). We aim to evaluate multiple combinations of operators and evaluate their impact on the resource utilisation and end-to-end latencies on decoupled scenarios.

This *decoupled* scenario (*i.e.*, using cloud and edge resources) is considered because it enables reducing the amount of data transferred at different phases, avoiding network restrictions that might exist at certain points of a path from edge to the cloud. Moreover, we are interested in evaluating the use of components traditionally employed for cluster/cloud-based stream processing solutions in more decentralised environments such as edge computing, where individual components may be hosted at a micro data centre or constrained resources geographically close to data sources whereas other services can run in the cloud.

3. Experimental Setup and Results

This section describes the environment setup and results of a primary evaluation to demonstrate the impact of deploying operators on cloud and edge resources.

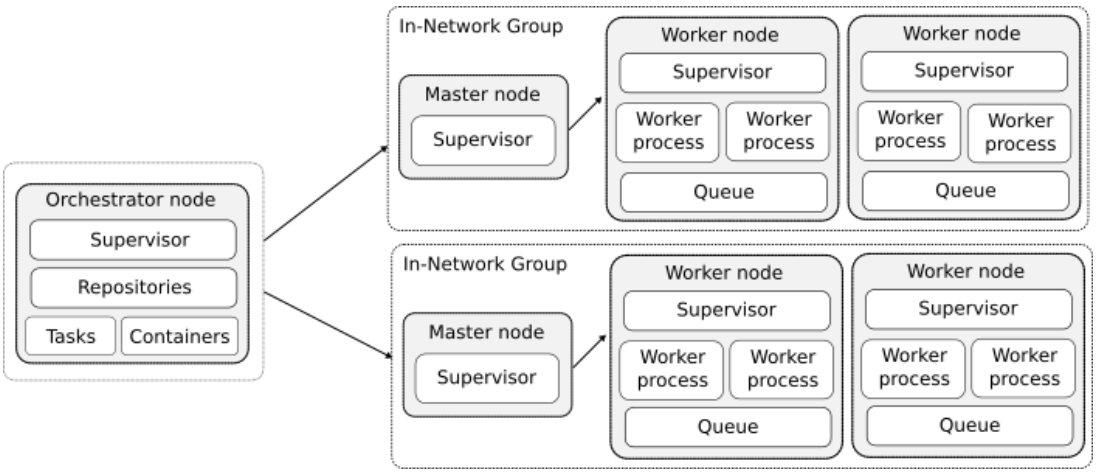


FIGURE 2 – Proposed architecture to improve resource utilisation and throughput.

3.1. Experimental Setup

The experiments comprise empirical evaluation performed on a cluster with four R410 Dell servers (Intel® Xeon® Processor E5506 4M Cache, 2.13 GHz, 4.80 GT/s Intel® QPI). The clock of all hosts are synchronised using Network Time Protocol (NTP). The model presented in the physical infrastructure, Figure 4, introduces the following roles:

— **Data Sources**: generate data and send it to the message queues. The data is drawn from an internal dataset with 1 GB of tweets that is processed by a sentiment analysis application described later. The tweets are sent by a built-in-house application that receives several parameters (*i.e.*, inter arrival time between tweets and number of processes) to deploy and stress the infrastructure. For this task we use one R410 Dell server.

— **Gateway**: receives the tweets and either treats them or forwards them to a message broker (*i.e.*, *bypass*). In other words, when one or more operators are deployed in the edge, the messages are stored in a lightweight queue (*i.e.*, Mosquitto 1.4.11) and processed by a lightweight DSPE (*i.e.*, Apache Edgent 1.0.0). Otherwise, the tweets are forwarded to the Cloud without any edge processing. In the physical infrastructure, this represents one R410 Dell server.

— **Cloud**: receives the messages in two distinct queues (*i.e.*, Apache Kafka 0.10.2): (i) sink, if the messages were processed by the Gateway ; or (ii) source, tweets which need some treatment. The DSPE (*i.e.*, Apache Flink 1.2.0) in the Cloud will treat the messages from the source queue. The DSPE was set up with two workers, it represents two R410 Dell servers (*i.e.*, one for the Flink Manager and Flink Worker ; and another for Flink Worker).

A sentiment analysis application that evaluates the polarity of tweets was used for performance evaluation (Figure 3). It uses a simple Natural Language Processing (NLP) technique to indicate the polarity of a sentence (*i.e.*, counting positive and negative words and computing the difference). The tweets are JSON dictionaries, each tweet corresponding to an event. Each event is parsed to extract the relevant fields (*e.g.*, tweet ID, language and the message itself). Then, the events are filtered by language, keeping only those that are in English. Next, the Stemmer removes stop words which do not carry sentiment or are irrelevant for the following steps. After that, an operator counts the number of negative and positive words, thus creating positive and the negative scores. At last, the application determines whether the tweet is positive or negative.

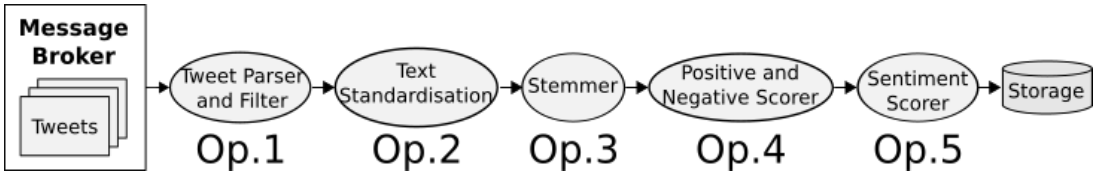


FIGURE 3 – Operators of the sentiment analysis application.

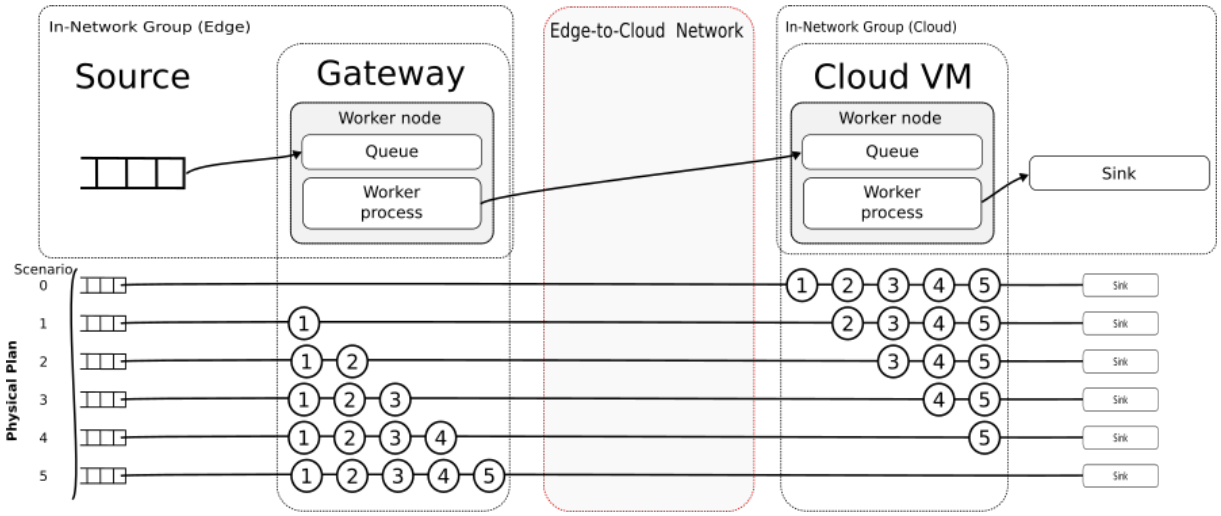


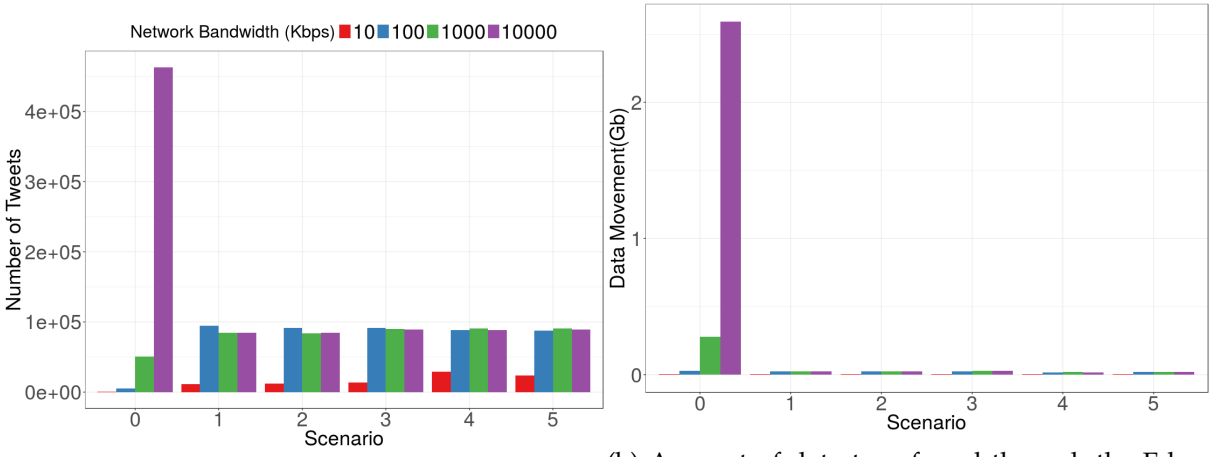
FIGURE 4 – Physical infrastructure and physical plan for deployment scenarios.

The configuration of the different stream-processing flows is presented in Figure 4. As presented in the physical plan, the stream operators will be deployed on the physical network. Scenario 0 represents all operators deployed in the cloud. In this way the *Data Source* will forward the messages directly to the Cloud, so that the Gateway will play the role of a *bypass*. Otherwise, under Scenario 6, all operators are deployed in the Gateway, and the Cloud acts as the *sink*. The remaining scenarios have mix configurations where operators are partially deployed in the Cloud and the Edge.

To evaluate the proposed environments and assess the impact of the network bandwidth, the following tools were used: Linux Traffic Control to customise the network bandwidth ; Python psutil 5.2.0 used for measuring resource consumption, CPU utilisation, memory usage and network I/O. Each individual scenario evaluated is performed in 7 minutes considering the testbed specification. We disregard the first and last minutes of the experiment to eliminate warm up and cool down effects. Each experiment corresponds to the deployment of the operators with a determined network bandwidth (*i.e.*, 10, 100, 1000 and 10000 Kbps) capacity on the edge-to-cloud network.

3.2. Experimental Results

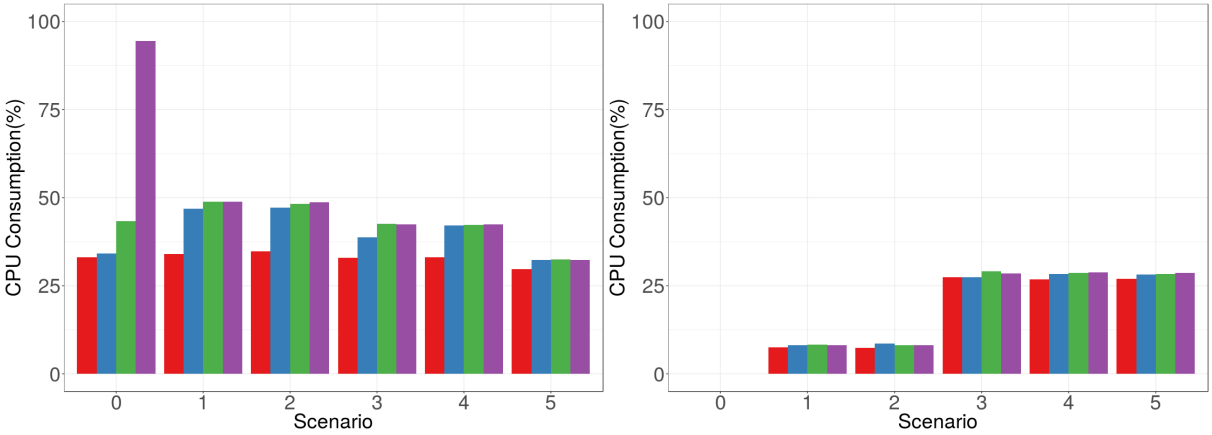
We noticed that the variations in network bandwidth and the operator placement have a direct impact on the number of treated events. This problem is evident when the edge-to-cloud network capacity is not enough (10, 100 and 1000 Kbps) to transfer the amount of data as presented in Figure 5a and 5b. However, to overcome this problem, we deploy the operators along the path, more specifically at the Gateway (source) and the cloud (sink). As depicted in Figure



(a) Number of tweets processed.

(b) Amount of data transferred through the Edge-to-Cloud network.

FIGURE 5 – Amount of tweets processed and through the Edge-to-Cloud network.



(a) CPU usage of the Cloud.

(b) CPU usage of Gateway.

FIGURE 6 – CPU usage comparison.

1, by placing operators in the edge we achieve the best solution. These results are obtained without much load on the Gateway as it just used of 28% of the CPU in average (Figure 6b) when the scenario 5 is considered and the edge-to-cloud network is limited to 1000 Kbps. Also, this usage limitation respects the edge constraints as edge devices have less computational power than the cloud.

In contrast, the results change substantially when the network bandwidth is not restrictive. As depicted in Figure 5a, when the bandwidth capacity is greater than 1000 Kbps, the edge becomes an execution bottleneck because the application is exploiting only one gateway that runs a lightweight stream processing framework that does not have all the full-fledged features and parallelism provided by Flink, used in the cloud. Although we performed experiments considering only one gateway, we argue that this is not the most practical scenario. In actual deployments, we expect scenarios that: contain multiple edge resources and several gateways (as in Figure 1) that can be used to offload some processing tasks from the cloud, or multiple paths that have several gateways between the sources and the sink.

The results allowed us to understand better the impact of the placement of data streams ope-

rators and the benefits that certain placement configurations can bring. The benefits concern optimising the number of events processed, the resource consumption and the monetary cost reduction caused by transferring less data to the cloud. Moreover, the lower utilisation of cloud resources as depicted in Figure 6a could be exploited to release unused capacity via auto-scaling operations and hence result in further lower costs.

4. Related Work

Over the years several frameworks for distributed data stream processing have been proposed, such as Apache Storm, Apache Spark [14] and Apache Flink [4]. In many of these frameworks, the applications are structured as directed graphs of operators that execute either pre-defined functions such as filtering, joins and splitting, or user defined functions. Most solutions are designed to run in homogeneous clusters, but have also been deployed in cloud environments. More recently, services are increasingly being employing on environments that span multiple data centres or on the edges of the Internet (*i.e.*, edge and fog computing). Existing work proposes architectures that place certain stream processing elements on micro data centres closer to where the data is generated [6] or that employ mobile devices for stream processing [10]. Even though these efforts are important, the present work focuses on evaluating the data-source-to-cloud latency when using software frameworks that are traditionally deployed for on cloud environments for more decoupled environments.
Most of the existing proposed work [8, 13, 15, 9] focuses on environments that do not consider variations in network bandwidth. The present work in contrast, takes into consideration both infrastructure and application constraints. We intend to focus on highly dynamic environments, where adapting an execution plan (the execution graph) is necessary to reduce to optimise the use of cloud and edge resources, and optimise the application end-to-end latency.

5. Conclusions

In this work we evaluated the impact in terms of number of tweets handled by a stream processing application when varying the network bandwidth and the deployment of operators across cloud and edge Computing resources. We observe that for the considered application, the partial deployment of operators between the infrastructures brings some important benefits. When a data stream flows through the operators, generally its data size becomes smaller, and depending on the network bandwidth, the edge deployment improves the number of processed events.

Bibliographie

1. Apache ActiveMQ. – `http://activemq.apache.org/`.
2. Apache Kafka. – `http://kafka.apache.org/`.
3. RabbitMQ. – `https://www.rabbitmq.com/`.
4. Alexandrov (A.), Bergmann (R.), Ewen (S.), Freytag (J.), Hueske (F.), Heise (A.), Kao (O.), Leich (M.), Leser (U.), Markl (V.), Naumann (F.), Peters (M.), Rheinländer (A.), Sax (M. J.), Schelter (S.), Höger (M.), Tzoumas (K.) et Warneke (D.). – The Stratosphere platform for big data analytics. *VLBD Journal*, vol. 23, n6, 2014, pp. 939–964.
5. Atzori (L.), Iera (A.) et Morabito (G.). – The internet of things: A survey. *Computer Networks*, vol. 54, n15, 2010, pp. 2787–2805.
6. Cardellini (V.), Grassi (V.), Presti (F. L.) et Nardelli (M.). – Distributed QoS-aware sche-

duling in Storm. – In *9th ACM International Conference on Distributed Event-Based Systems*, *DEBS '15*, DEBS '15, pp. 344–347, New York, USA, 2015. ACM.

7. Chan (S.). – Apache quarks, watson, and streaming analytics: Saving the world, one smart sprinkler at a time. – Bluemix Blog, June 2016.
8. Cheng (B.), Papageorgiou (A.) et Bauer (M.). – Geelytics: Enabling on-demand edge analytics over scoped data sources. – In *IEEE International Congress on Big Data (BigData Congress)*, pp. 101–108, June 2016.
9. Hochreiner (C.), Vogler (M.), Waibel (P.) et Dustdar (S.). – VISP: An ecosystem for elastic data stream processing for the internet of things. – In *20th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2016)*, pp. 1–11, Sept 2016.
10. Morales (J.), Rosas (E.) et Hidalgo (N.). – Symbiosis: Sharing Mobile Resources for Stream Processing. – In *IEEE Symposium on Computers and Communications (ISCC 2014) – Workshops*, pp. 1–6, June 2014.
11. Tziritas (N.), Loukopoulos (T.), Khan (S. U.), Xu (C. Z.) et Zomaya (A. Y.). – On Improving Constrained Single and Group Operator Placement Using Evictions in Big Data Environments. *IEEE Transactions on Services Computing*, vol. 9, n5, September 2016, pp. 818–831.
12. Uckelmann (D.), Harrison (M.) et Michahelles (F.). – An architectural approach towards the future internet of things. *In : Architecting the internet of things*, pp. 1–24. – Springer, 2011.
13. Wu (Y.) et Tan (K. L.). – ChronoStream: Elastic stateful stream computation in the cloud. – In *2015 IEEE 31st International Conference on Data Engineering*, pp. 723–734, April 2015.
14. Zaharia (M.), Chowdhury (M.), Das (T.), Dave (A.), Ma (J.), McCauley (M.), Franklin (M. J.), Shenker (S.) et Stoica (I.). – Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. – In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, *NSDI'12*, NSDI'12, pp. 2–2, Berkeley, CA, USA, 2012. USENIX Association.
15. Zeng (D.), Gu (L.) et Guo (S.). – *A General Communication Cost Optimization Framework for Big Data Stream Processing in Geo-Distributed Data Centers*, pp. 79–100. – Cham, Springer International Publishing, 2015.