University of Nice Sophia Antipolis

Ecole Polytech Nice-Sophia

Projet de Fin d'Études 2019-2020

Big Data Stream Processing in Multi-Source Cloud / Fog Environments

at

i3s Laboratory - Sophia Antipolis, France

Student:
Abdelkarim HAFID

Supervisors:
Fabrice Huet
Alessio Pagliari

Contents

Abstract

Data streaming is the continuous transfer of data at a steady and high-speed rate. It applies well to data sources that send data in continuous flows. Among these sources, I can cite: telemetry operations from connected devices, log files generated by users during their various journeys on an application or website, e-commerce transactions, or any information from social networks. Today, A drilling tower generates 500 gigabytes of data per week. An airliner's turbine provides 10 terabytes in 30 minutes and The number of IoT applications has increased exponentially.However, The using of the Cloud to stream all the data generated is inefficient and very expensive, because the cloud-based architectures often centralize storage and processing, which generates high data travel costs that penalize real-time applications. It is therefore necessary to decide on the edge of the network which generated information must be transferred to centralized systems to be recorded there, and which data can be interpreted on the edge, the fact that led to the emergence of the Fog architecture as a way of distributing treatments from centralized clouds to decentralized processing near to data sources. Moreover, The Fog architecture is reducing the cost of data movement and improving the response time and the throughput of the data streaming. In particular, we will try to prove by one experimentation that you find in the chapter 4 of my report on what extent we can improve or degrade the performance of an application by performing calculations closer to the data rather than in the Cloud. We have analysed two performance metrics: the throughput and the latency, in order to understand the scenarios who could benefit from the hybrid FoG+Cloud architecture and the scenarios that must be executed in the cloud only.

Chapter 1

Introduction

Data streaming allows analysis and monitoring of data in real time and in a countinous manner,the use of data streaming is optimal when duration is a major dimension of the data. We can think for example of tracking the duration of a web session. Overall, most of the data from the Internet of Things (IoT) is compatible with data streaming: traffic sensors, transaction or activity logs which allows information to be generated on a wide range of activities. For instance : counting activity, server activity, device geolocation and clicks on a site web.The conventional approach to implementing these IoT applications is to send data from all sources to the cloud and leverage its powerful resources to process incoming data flows and perform analytics. However, this approach quickly becomes unsustainable due to the impact on latency, network congestion, storage costs, the throughput, the volatility and the privacy of data.

1.1 Motivations

Cloud computing systems offer customers low-cost services mainly in terms of storage and computing. However, the centralized architecture of the cloud can exhaust the available bandwidth and increase the service time. For instance, the large distance between the source of data and where the processing is done results in high service latency, which is not recommended for time-sensitive security applications such as healthcare, manufacturing and smart cities. Some streaming applications have huge amounts of generated data collected from sensors that could exhaust network bandwidth, so traditional cloud systems are not sufficient to meet the requirements of these applications due to the great distance between the source of data for example IoT sensors and the processing elements at the heart of the cloud.The model of the fog computing will resolve this issues by minimizing the distance between the generation and the processing of the data and by providing the service support to the geographically separated and real-time services. The streaming of applications in the Fog environment take place to extend the streaming of applications in the cloud by serving some customers' requests by devices located at the FoG instead of sending them to be executed at the cloud core.

1.2 Objectives

Fog computing appears recently as a computing model to face and resolve the issues of the Cloud computing systems. The key idea is to leverage computing and storage resources at the FoG of the network. This allows applications to outsource data processing from the main (Cloud) processing data centers to the FoG. So, our ultime goal of this PFE is to see if the Stream Processing in the FoG will allow the reduction of the latency and will improve the throughput of the streaming of the applications. Furthermore, I will define the way of working and the components of the plateform of streaming Storm because we will use it in the experimentation's to prove that streaming in the FoG+Cloud will increase the performance metrics. Moreover, we will try to study how we can deploy the hybrid FoG+Cloud architecture to have an online and real-time front-end for processing on the Fog 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 or with tasks that overlap the capacity of the FoG.

1.3 Challenges

Data streaming in the FoG+Cloud architecture remain the crucial solution to deal with the issues of the streaming in the Cloud only, but it still have a lot of challenges specifically in the FoG like node volatility, limited processing power, high latency between nodes, fault tolerance and data degradation the thing that may impact the performance of the applications. The use of the Fog+cloud architecture in the data streaming poses the following challenges:

- 1) Deciding how to split the streaming of applications among the FoG and Cloud resources
- 2) Focusing on the challenge of the failure of nodes that restrict the streaming of applications in the Fog architecture because of the great dynamic and heterogeneity nature of the devices and the interactions between them. [1]
- 3) Moving operators from cloud to edge to stream the applications will include devices limitations with respect to memory,CPU,and often network bandwidth.

Chapter 2

Definition of the Data Stream Processing in multi-Source Cloud/Fog environments

A data stream is an unbounded and continuous flow of data. Examples include data produced by sensors or user events from social networks such as comments,postings and tweets. Stream processing applications typically collect data from the Edge of the networks, where they are produced and pre-processed and send the results to the Cloud that is used as centralized architecture where the data are shipped for further, complex analytics by using the batch processing.

2.1 An overview of the Streaming Platform Storm

Storm is an open source, fault-tolerant, distributed and reliable system for processing streams of data. A Storm cluster has three sets of nodes:

1. **Master node:** runs a deamon called "Nimbus", which is responsible for distributing the tasks across the cluster, assigning tasks to the machines and monitoring for failures.

2. **Worker nodes:** run a daemon, called the "Supervisor",that receives the tasks assigned by the master node, and starts and stops the worker processes as dictated by "Nimbus" on the master node.

3. **Zookeeper nodes:** coordinate the communication between master and worker nodes . The overall size of the ZooKeeper cluster will depend on the Scale of the Storm cluster, and the level of resilience desired.

A streaming application in Storm is called topology. There are two types of processing components in Storm: "spouts" and "bolts". A spout is the source of a data stream and it is the one who emits the data, while a bolt is the computational unit used to process the data before emitting new data to the next bolt in the DAG(Direct Acyclic Graph).For instance, in my experimentation,I have used a linear DAG with 1 Spout and 3 bolts to stream the generator prototype application NAMB. Each component in Storm consists of a number of executors that can be run in parallel and each executor normally consists of one task. When a topology, consisting of Spouts

and bolts, is submitted to a Storm cluster, the tasks are grouped into a number of workers. Each compute node is configured with a number of slots where each worker is assigned to one slot.

2.2 Data Stream processing 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 2.1: **"Ingestion"** and **"processing"**.

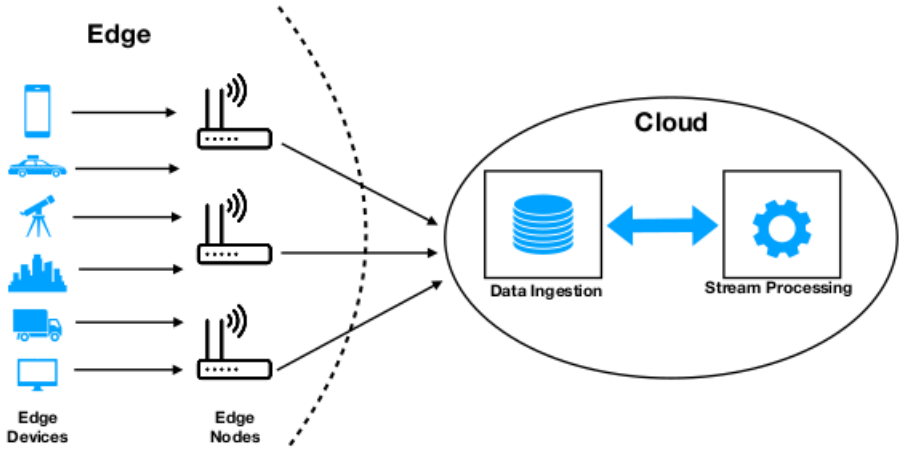


Figure 2.1: Typical stream computing from the Edge to the Cloud involving data ingestion and stream processing. Source:[2]

- 1)Data ingestion: this step serves to log, aggregate and buffer data streams before they are consumed by the processing phase. The Apache Kafka is the well known and the most widely used tool that ensure the Data ingestion.
- 2)Stream processing: executes data analytics on streams. Normally, the streams must be processed in real-time with a high throughput and low-latency so a large set of stream processing engines were proposed specifically to address these challenges. The most notable ones are Flink , Spark , and Storm.

2.3 Data Stream processing in the fog+cloud environments

Edge computing is the collection of technologies that allows computation to be performed at the Edge of a network [4]. 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 in order to reduce latency and transmission costs [5].The performance of image and video processing on mobile devices is improved by offloading part of the computation to Edge servers [6].In all this related works, The events emitted by sensors are processed locally in order to reduce latency , transmission costs and to increase the throughput. Moreover, Fog

computing: is a paradigm similar to the Edge computing because it enables computations 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 the network gateways. So, to stream applications in the fog+Cloud environments we need to analyse the impact of some parameters on the performance of the streaming: For instance, the impact of the bandwidth on the Edge-to-Cloud throughput and the impact of the bandwidth on the end-to-end latency.

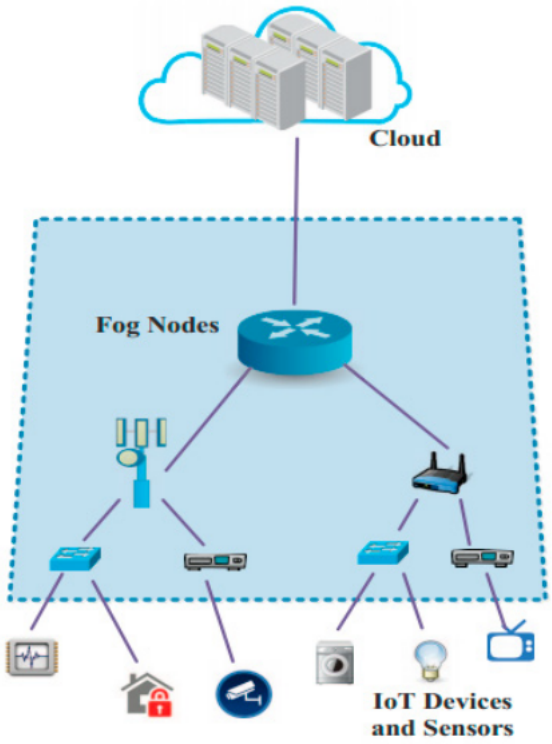


Figure 2.2: Fog architecture.[3]

Figure 2.2 Illustrates a simple structure for fog computing. The architecture consist of three layers named: End-clients,Fog layer, and Cloud layer.The third layer is Cloud Layer, cloud server are generally unsuccessful to manage storage and preparing requests of billions of IoT devices,they suffered from congested network, low response time in service delivery, low quality of Service.Generally,a fog computing domain is made out of normal network devices:routers,switches,servers,Base Stations,and it can be deployed closer to IoT sensors.Furthermore,the data streaming on the fog computing can perform effectively in term of the throughput and the network traffic, content dissemination, power conservation, and latency because the devices and sensors are everywhere,and more are coming online every day. So the target of scientists when they stream applications is to Reduce the amount of data transmitted to the analytics servers and also reduce the amount of data stored. In this way, the concept of Fog/edge computing in the field of data streaming is become the crucial solution enabled by several platforms that target software deployment, data processing and system monitoring.For instance, the technology of **Apache Edgent** has appeared

like a light-weight open source Java library that can be easily deployed on edge devices and nodes. Normally to do data streaming on the fog architecture or on the FoG+Cloud we need applications that uses analytics to determine when data needs to be sent to a back-end system for further analysis, action, or storage. For example, we can use Edgent to determine whether a system is running outside of normal parameters.If the system is running normally, you don't need to send this data to be streamed on the back-end system (The Cloud), it's an added cost and an additional load on your system to process and store it in the Cloud. However, if Edgent detects an issue in the streaming of one application, in this time it will transmit that data to the back-end system to determine why the issue is occurring and how to resolve it. To do data streaming in the Fog+Cloud environment we need to limit the quantity of data sent to the Cloud because the cost of communication is high and the bandwidth is limited. For instance,We can send data to our Cloud when we need to perform analysis that cannot be performed on the edge device, such as:

- Running a complex analytic algorithm that requires more resources, such as CPU or memory, than are available on the edge device.
- Correlating data from the device with data from other sources, such as: Weather data, Social media data, Data from other devices

The technologies used by the Fog to communicates with the back-end systems when they stream an application are listed here:

- 1)MQTT – The messaging standard for IoT: is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. For example, it has been used in sensors communicating to a broker via satellite link. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers
- 2)IBM Watson is a cloud-based service that provides a device model on top of MQTT : It is a managed by the cloud and it is designed to easily leverage the IoT devices. It offers many features: device registration, connectivity, control, rapid visualization and storage of Internet of Things data.
- 3)Apache Kafka: is a streaming platform that has three key capabilities:1)Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.2)Store streams of records in a fault-tolerant durable way.3)Process streams of records as they occur. Kafka is generally used for two broad classes of applications:1)Building real-time streaming data pipelines that reliably get data between systems or applications 2)Building real-time streaming applications that transform or react to the streams of data

.

Chapter 3

Use cases of the Data Stream Processing in the multi-Source Cloud / Fog Environments

3.1 PrEstoCloud Project

3.1.1 Introduction to the PrEstoCloud Project

The project of PrEstoCloud is one of the main use cases of the datastream processing in the multi-source Fog+Cloud environments and it has been initiated to point out the 3 following goals:

- exploit multi-cloud environments for deploying Big Data processing frameworks
- make intelligent cloud placements and configurations of applications based on the anticipated processing load with respect to data volume and velocity (Speed of data)
- elaborate on components that are capable to recommend and implement adaptations in real-time

PrEstoCloud covers, exactly, these challenges and limitations, contributing to the evolution of real-time Big Data processing [7].

3.1.2 Logistics Use case - CVS

In recent years, the use of different sensors connected to vehicles is challenging, because the major challenges are not in the collection and storage of data, but also in the fast extraction of useful information and the triggering of alerts to help decision makers : "driver, logistic center , vehicle owner or insurance company" perform right decision at the right time . To this end , the data generated by vehicles sensors need to be processed at run-time to recognise important driving actions in an instant.In the CVS logistic use case, edge nodes deployed in vehicles are employed to analyse data at run-time next to the location where the sensory data are generated. In this way, if the edge node cannot appropriately execute computing operations because it is overloaded due to receiving one more video data streaming generated by an extra camera switched

on, computing tasks provided on the edge node should be terminated and started on the cloud infrastructure.

In the CVS logistic use case, different types of data will be analysed in real-time to provide timely decisions and make useful suggestions. Data analytics will be carried out on whether edge or cloud, dependent on the situation at run-time.The sensor data is collected by the edge node located in the vehicle. The results(alerts and suggestions) are sent back to the vehicle reported through the driver's tablet or mobile phone in real-time or semi-real-time.To achieve this vision, the key drawback of the current CVS Mobile fleet management system is its inability to provide real-time feedback to the driver and other entities such us logistic centre to make timely decisions. This is because timely decisions will lead to the optimizations on different levels, such as driving style or routing towards a destination. The main reasons for this drawback are:

- lack of sensory information to perform certain type of decisions. In particular, for driving style suggestions, there is a need for sensors such as compass, gyroscope, and accelerometer. To monitor the environnement around the vehicle, there is a need for video cameras.

- Lack of real-time analytics accomplished on the incoming streaming data. Some of the analytics can be done on the edge, and some more complex tasks which need more computing capacity can be performed on the Cloud.

All the features performed by the driver can be analysed and they allow to form a profile which represents the driver's behaviour. Driver's profiles is able to show useful information on how safe their driving could be, how economic they drive in terms of fuel-consumption, how much they care about vehicle maintenance, how efficient their driving is in terms of environmental impact.

3.1.3 Wide Area Video Surveillance cameras use-case : "Videostone technology"

Current Surveillance systems in Enterprise facilities and public places produce massive amounts of video content while operating at a 24/7 mode. There is an increasing need to process such huge video data streams to enable a quick streaming of events that are happening during a specified time frame in a particular location . Concepts like fog computing based on localisation of data processing will relax the need of existing Cloud-based solutions. To address the current threats of security and make the existing cameras more efficient they decided in the PrestoCloud project to deploy the Videostone technology like a direct application of the streaming in FoG+Cloud environment . Videostone is designing a new system that will cover all senstive areas including public administration buildings, bus station,shopping malls and main squares.

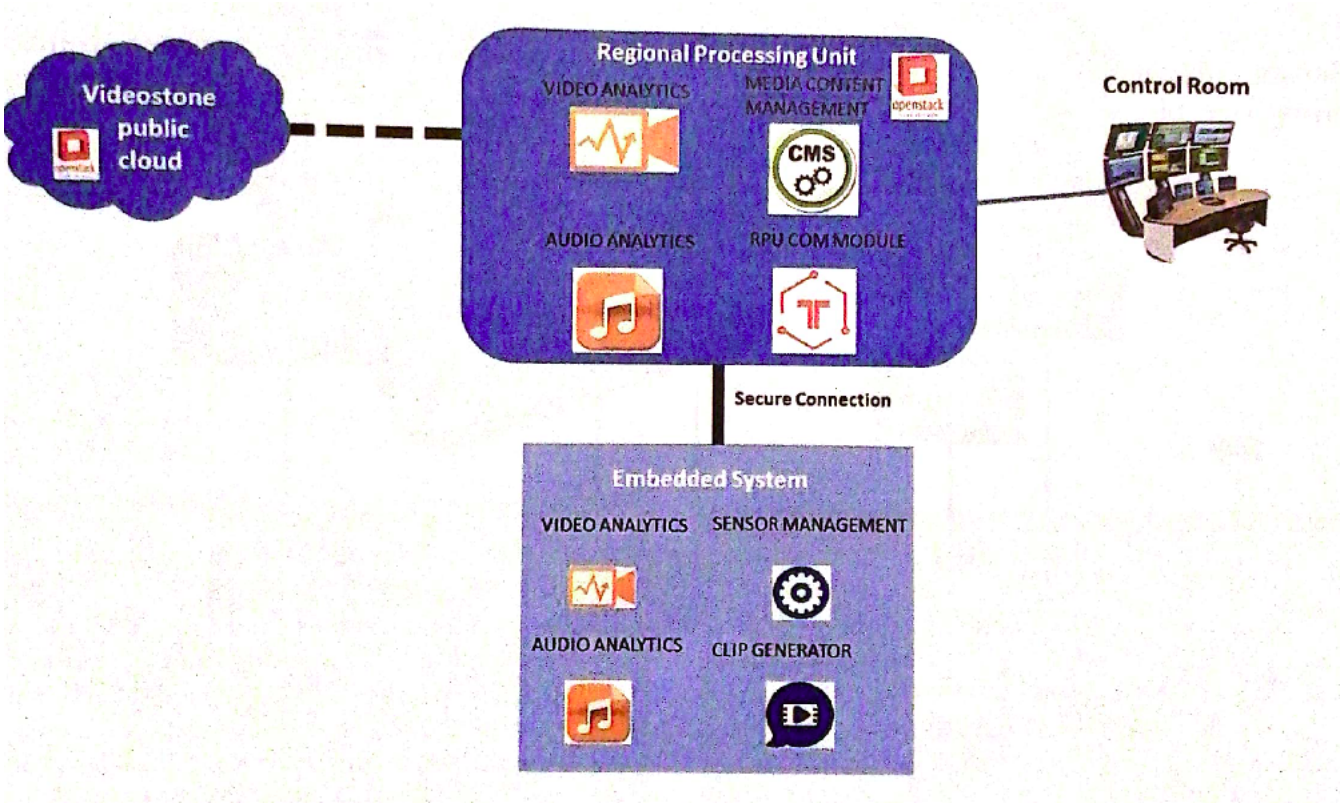


Figure 3.1: Videostone technology.[8]

As the picture above illustrate, The Regional processing unit will be connected to the public Videostone, when needed, to make use of additional computing resources. In each one Regional Processing Units, we found the video walls where security personnel can watch on real time the video streams coming from the cameras. In the case of an identified threat, an alarm is triggered and security personnel notify the Municipal Police for response. Furthermore,Videostone engineers suggested that services synchronisation and migration should take place in three different layers, as shown in the previous image :

- **Camera-build: (ES)embedded system** The ES runs "light" video and audio analytics that require low processing power. When any type of the security threats is detected, then the video is streamed back from the camera to the respective Regional processing units for further identification and verification.

- **RPUs:Regional Processing Units** At this level, all RPUs will operate under a common monitoring service where computing and network resources take place to do: Video Analytic,Audio Analytics, Versatile Media Content Management and to ensure RPU Communicator(RCOM) to receive the Clips from the Embedded Systems, de-encapsulate, forward the contained Clip Object to the Videostone PublicCloud and to handle the VPN connections directly between the RPU and the Embedded systems .

- **Videostone Public Cloud:** At this level service migration between RPUs and the Videostone cloud will be realised when RPUs are not able to handle sudden picks in the workload.

Videostone is seeking for solutions to increase both accuracy in terms of threat detection and efficiency in terms of response time, when handling a large amount of video streams. This novel architecture has mainly three advantages: 1) It will reduce the company's operational cost for migrating all services in a public cloud 2) It can provide better response time in terms of threat detection and identification and 3) It will preserve data privacy rules as the data with personal information will not be located outside of the country.

3.2 New York city TAXI and Limousine Commission (TLC)

In this use case they analyse information about taxi rides such as driver identification, pick-up and drop-off times, and locations [9].The scenario they implemented consists in finding the busiest driver every two hours. To do that, five operations need to be performed: the first one is to filter records with invalid data; The second one is to filter records that is defined like the percentage of data filtered out of the stream.The third step is to pickup and drop-off positions and times that will be used to calculate the travel distance and average speed;in the fourth step, the windowed aggregation gets the sum of working times for each driver in the last two hours and in the last step, 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 and the operations (4) and (5) are performed on the Cloud.

3.3 OBSERVE ORIENT DECIDE ACT LOOP : OODA use case

The Observe Orient Decide Act (OODA) loop refers to the decision-making cycle of observe, orient, decide, and act, developed by military strategists and the United States Air Force [10]. OODA is a decision-making cycle to process data streaming from sensors in real time and it become recently an essential design characteristic for IoT applications. Anshu et al. [11] offer a suite of IoT applications that follows the closed-loop OODA cycle. The applications are based on common IoT patterns for data pre-processing, statistical summarization, and predictive analytics.

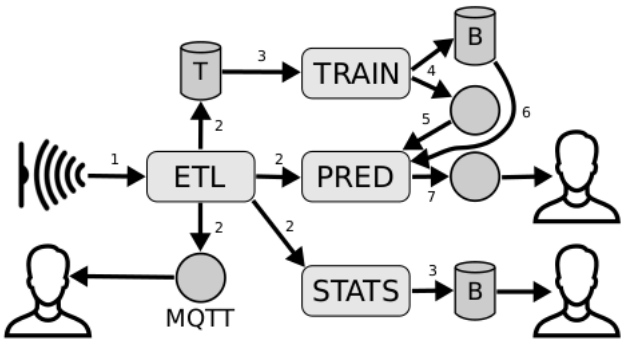


Figure 3.2: RIoTBench IoT high-level logical interactions between different sensors, applications and users. Source : [12]

the Extract-Transform-Load (ETL) consumes data from hundreds of thousands of edge sensors, and preprocesses, cleans, and archives the data. Further, the results are published to an edge broker so that clients interested in real-time monitoring can subscribe to it, while a copy is forked to the cloud for storage, and another to the next dataflow step.The ETL dataflow requires a low-latency cycle in order to achieve real-time monitoring, in addition it also requires some of its operators to be located in the cloud for storing messages and others to be at the edge of the network.The first step in the OODA loop use case is to do some **Statistical Summarization (STATS)** to perform higher order aggregation and plotting operations, and stores the generated plots into the cloud, from where webpages can load the visualization files on browsers,and the second step is the **Model Training (TRAIN)** that periodically loads the stored data from ETL step and trains forecasting models that are stored in the cloud, and notifies the message broker of an updated model being available and the last step is **The Predictive Analytics (PRED)** that subscribe to the message broker and downloads the new models from the cloud, and continuously operates over the pre-processed data stream from ETL to make predictions and classifications that can indicate actions to be taken on the domain. It then notifies the message broker of the predictions, which can independently be subscribed to by a user or device for action.

Chapter 4

An overview of the experimentation's carried out in this PFE to prove the advantages and the drawbacks of the FoG+Cloud architecture

4.1 Introduction

In my technical work, I worked with Storm and one prototype application generator called NAMB. I worked with the Grid'5000 as a large-scale and flexible test bench for experimental research in all areas of IT in order to reserve nodes where I can do my experiments in the clusters of Sophia and Rennes. I select one cluster in Sophia named 'Suno' with these characteristics: "the architecture of the CPU is 2 x Intel Xeon E5520, it has 4 cores/CPU and 32 GB of memory" and another cluster in Rennes named Parapluie with these characteristics: "the architecture of the CPU is 2 x AMD Opteron 6164 HE, it has 12 cores/CPU and 48 GiB of memory". Given these architectures, we have selected the cluster Parapluie in Rennes to play the role of the Cloud because it is powerful :12 cores/CPU and the Suno Cluster of Sophia to play the role of the FOG because it has only 4 cores/CPU. So, for the first time, I installed Storm in the FOG + Cloud environment and I runned the prototype application generator NAMB on Storm installed in this multicluster. After that, I installed Storm only on the cloud in the Parapluie cluster in Rennes and I executed NAMB in this cluster. Our objective from these experiences is to see the impact of joining the FOG Suno in Sophia to the Cloud Parapluie in Rennes on the performance of the streaming of the prototype NAMB over Storm . We will focus on two metrics latency and throughput measurements to see the performance of the FOG + Cloud architecture.

4.2 Deployment of Storm in the Fog+Cloud environment and the Streaming of the prototype application generator NAMB over Storm

In the First time, I will connect my local machine to the Grid5000 and specifically to the serveur of Sophia : 1) ssh ahafid@access.grid5000.fr 2) ssh sophia

```
(base) user@user-Latitude-E5420:~$ ssh ahafid@access.grid5000.fr
Linux access-north 4.9.0-8-amd64 #1 SMP Debian 4.9.144-3 (2019-02-02) x86_64
----- Grid'5000 - access-north.grid5000.fr -----

Welcome to Grid'5000

** Connect to a site:
$ ssh {grenoble,luxembourg,lyon,nancy,nantes,rennes,sophia}

** Useful links:
 - account management (password change): https://api.grid5000.fr/ui/account
 - homepage: https://www.grid5000.fr/mediawiki/index.php/Category:Portal:User
 - charter : https://www.grid5000.fr/mediawiki/index.php/Grid5000:UserCharter
 - support : https://www.grid5000.fr/mediawiki/index.php/Support

** Data on access.grid5000.fr :
 - your home directory on access machines (access-north and access-south)
   is not synchronized and should not be use to store data.
 - please use ssh forwarding to send data directly to sites or
 - (outside) $ scp files login@access.grid5000.fr:reims/ using the nfs mount point in your home
Last login: Tue Jan 14 14:09:44 2020 from 134.59.130.125
ahafid@access-north:~$ ssh sophia
Linux fsophia 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64
----- Grid'5000 - Sophia - fsophia.sophia.grid5000.fr -----

This site has 2 clusters (see: https://www.grid5000.fr/w/Sophia:Hardware)

Available in queue default:
 - suno (2010): 45 nodes (2 CPUs Intel Xeon E5520, 4 cores/CPU, 32GB RAM, 557GB HDD, 1 x 1Gb Ethernet)
 - uvb  (2011): 43 nodes (2 CPUs Intel Xeon X5670, 6 cores/CPU, 96GB RAM, 232GB HDD, 1 x 1Gb Ethernet, 1 x 40Gb InfiniBand)

** Useful links:
 - account management (password change): https://api.grid5000.fr/ui/account
 - homepage: https://www.grid5000.fr/mediawiki/index.php/Category:Portal:User
 - charter : https://www.grid5000.fr/mediawiki/index.php/Grid5000:UserCharter
 - support : https://www.grid5000.fr/mediawiki/index.php/Support

** Others sites:
$ ssh {grenoble,lille,luxembourg,lyon,nancy,nantes,rennes}

** 1 bug could affect your experiment (see https://www.grid5000.fr/status/artifact/)
--> #Sophia poor performance of infiniband on some uvb nodes
    https://intranet.grid5000.fr/bugzilla/show_bug.cgi?id=10813

Last login: Tue Jan 14 15:30:51 2020 from 192.168.66.33
ahafid@fsophia:~$
```

After that, I will reserve 4 nodes in 'Suno' and 4 nodes in 'Parapulie' in the same time to deploy Storm in this nodes simultaneously.

```
ahafid@fsophia:~$ oargridsub -t deploy -w '2:30:00' suno:rdef="/nodes=4",parapluie:rdef="/nodes=4"
suno:rdef=/nodes=4,parapluie:rdef=/nodes=4
[OAR_GRIDSUB] [parapluie] Date/TZ adjustment: 0 seconds
[OAR_GRIDSUB] [parapluie] Reservation success on parapluie : batchId = 1243844
[OAR_GRIDSUB] [suno] Date/TZ adjustment: 0 seconds
[OAR_GRIDSUB] [suno] Reservation success on suno : batchId = 927594
[OAR_GRIDSUB] Grid reservation id = 66897
[OAR_GRIDSUB] SSH KEY : /tmp/oargrid/oargrid_ssh_key_ahafid_66897
       You can use this key to connect directly to your OAR nodes with the oar user.
```

After the reservation of nodes, I configured the clusters 'Suno'+'Parapluie'. The picture below show the modifications that I did to adapt the file of configuration. First, I changed the user name to 'ahafid' and the deploy.image.name to debian9-storm-base and the multi.cluster to 'yes'. Moreover, I specified the version of Storm to be 1.2.1 and the zookeeper.version to be 3.4.12.

```
[g5k]
user.name = ahafid
deploy.image.name =debian9-storm-base
# node.memory.mb = 24576.0
node.memory.mb = 32768.0
node.cpu.units = 8
oar.file.location = ~/machines
multi.cluster = yes

[storm]
storm.version = 1.2.1
zookeeper.version= 3.4.12
csv.log.dir = ../storm_csv_metrics
csv.filter.expression = .*default.*emitted.*
# zookeeper.nodes = 1
nimbus.nodes = 1
workers.per.node = 20
workers.starting.slot = 6700
worker.max.heap.size.mb = 1024.0
worker.heap.memory.mb = 1024
storm.scheduler = org.apache.storm.scheduler.DefaultScheduler
# storm.scheduler = org.apache.storm.scheduler.resource.ResourceAwareScheduler

[ansible]
inventory.file.path = ./g5khosts
playbook.file.path = ./storm_playbook.yaml
~
```

After the configuration of the cluster, I will deploy Storm in the nodes reserved in Parapluie+Suno as shown in the two images below

```
ahafid@fsophia:~/g5k-storm-cluster-master$ python3 deploy.py

PLAY RECAP *********************************************************************************************************************************************
parapluie-11.rennes.grid5000.fr : ok=5    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-28.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-8.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-9.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
suno-2.sophia.grid5000.fr  : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
suno-25.sophia.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
suno-34.sophia.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
suno-8.sophia.grid5000.fr  : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Waiting startup...
curl 'http://parapluie-11.rennes.grid5000.fr:8080/api/v1/supervisor/summary'
curl: (7) Failed to connect to parapluie-11.rennes.grid5000.fr port 8080: Connection refused


**** ssh tunnel ***
ssh ahafid@access.grid5000.fr -N -L8080:parapluie-11.rennes.grid5000.fr:8080
ahafid@fsophia:~/g5k-storm-cluster-master$
```

now , Storm is installed on the 4 nodes in Sophia and the 4 nodes in Rennes simultaneously.After this step, I will Stream my application generator prototype NAMB in the Storm installed in the multicluster Sophia+Rennes. I selected the version 7.0.1 of the prototype application generator NAMB and I have added the file of configuration "namb.yml" to the repository of NAMB and after that I compiled the project in the cluster to be read to the execution over Storm. For the namb.yml file is shown in the picture below where I specified the structure of data that I will stream with Storm, by offering 100 to the size of the data and 100 to the value of the data. Furthermore, I specified the distribution of the flow to be uniform and I activate the reliability to enable "acking-like" mechanism.

```
datastream:

  synthetic:
    data:
      size: 100 # size of the single data
      values: 100 # different values
      distribution: uniform # [uniform, nonuniform]
    flow:
      distribution: uniform # [unifrom, burst]
      rate: 0 # msg/s: max value is 1000 (1 each ms), 0 is without pause between packages


##############################################
# WORKFLOW: application (DAG) properties
##############################################

workflow:

  ## DAG levels
  depth: 4

  scalability:
    parallelism: 12 # Total number of executors
    balancing: balanced # [balanced, increasing, decreasing, pyramid]

  connection:
    shape: linear #[linear, diamond, star]
    routing: balanced # [none, balanced, hash, broadcast]

  workload:
    processing: 3 # CPU load value in thousands of cycles
    balancing: balanced # [balanced, increasing, decreasing, pyramid]

  reliability: true # Boolean, true to enable "acking-like" mechanism, false otherwise

  filtering: 0 # percentage of filtered data [1: output=100% of input, 0.3: output=30% of input, and so on]

  windowing:
    enabled: false
    type: tumbling # [tumbling, sliding]
```

I have also adapted the file "storm-benchmark.conf" as shown in the picture below to sample the tuples to be printed on the log files , meaning it would print 1 every 200 tuples in the log files.

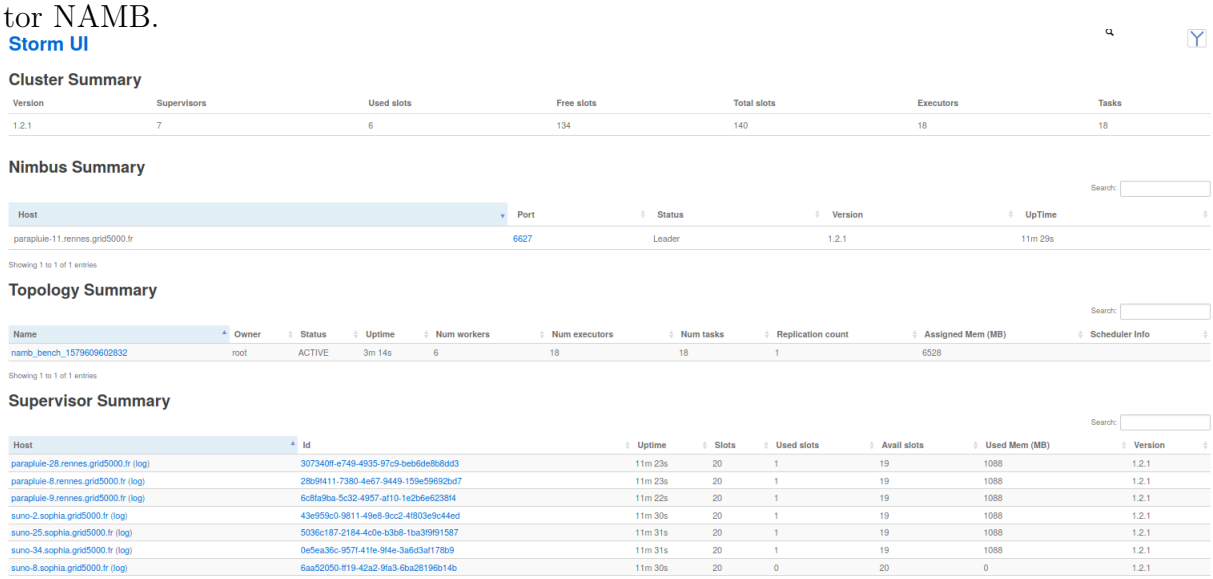
```
workers: 6
maxSpoutPending: 5000
deployment: cluster # local or cluster
debugFrequency: 0.005
~
~
```

The picture below prove that the application generator prototype NAMB is executed over Storm successfully:

```
ahafid@fsophia:~/namb-0.7.1$ python3 namb.py storm -p ~/apache-storm-1.2.1/bin/storm
Running: java -client -Ddaemon.name= -Dstorm.options= -Dstorm.home=/home/ahafid/apache-storm-1.2.1 -Dstorm.log.dir=/
ath=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file= -cp /home/ahafid/apache-storm-1.2.1/*:/home/ahafid/apa
1/extlib/*:/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar:/home/ahafid/apache-storm-1.2
jar=/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar -Dstorm.dependency.jars= -Dstorm.dep
plication /home/ahafid/namb-0.7.1/conf/namb.yml /home/ahafid/namb-0.7.1/conf/storm-benchmark.yml
spout_1 bolt_1 bolt_2 1063 [main] WARN  o.a.s.u.Utils - STORM-VERSION new 1.2.1 old null
1094 [main] INFO  o.a.s.StormSubmitter - Generated ZooKeeper secret payload for MD5-digest: -5535222314176892092:-52
1292 [main] INFO  o.a.s.u.NimbusClient - Found leader nimbus : parapluie-11.rennes.grid5000.fr:6627
1349 [main] INFO  o.a.s.s.a.AuthUtils - Got AutoCreds []
1397 [main] INFO  o.a.s.u.NimbusClient - Found leader nimbus : parapluie-11.rennes.grid5000.fr:6627
1444 [main] INFO  o.a.s.StormSubmitter - Uploading dependencies - jars...
1445 [main] INFO  o.a.s.StormSubmitter - Uploading dependencies - artifacts...
1445 [main] INFO  o.a.s.StormSubmitter - Dependency Blob keys - jars : [] / artifacts : []
1476 [main] INFO  o.a.s.StormSubmitter - Uploading topology jar /home/ahafid/namb-0.7.1/benchmarks/storm-bench/targe
apache-storm-1.2.1/local-dir/nimbus/inbox/stormjar-a7fcce4f-3f41-4d92-83cf-4bc018d933cf.jar
Start uploading file '/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar' to '/root/apache-
-3f41-4d92-83cf-4bc018d933cf.jar' (7040562 bytes)
[==================================================] 7040562 / 7040562
File '/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar' uploaded to '/root/apache-storm-1
d92-83cf-4bc018d933cf.jar' (7040562 bytes)
2248 [main] INFO  o.a.s.StormSubmitter - Successfully uploaded topology jar to assigned location: /root/apache-storm
-4d92-83cf-4bc018d933cf.jar
2249 [main] INFO  o.a.s.StormSubmitter - Submitting topology namb_bench_1579609602832 in distributed mode with conf
topology.auth.scheme":"digest","storm.zookeeper.topology.auth.payload":"-5535222314176892092:-5298774891732574914","
2249 [main] WARN  o.a.s.u.Utils - STORM-VERSION new 1.2.1 old 1.2.1
2833 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: namb_bench_1579609602832
```

now the application "NAMB" is running over Storm and the tuples are in the process of generation . After 10 minutes, I will Kill the application to Stop the generation of

the tuples and to have the ability to retrieve the log files from the nodes in Parapluie and Suno and analyse them.The results are shown in the image below, it prove the running of NAMB over Storm and show the nodes of the clusters Suno in Sophia and Parapluie in Rennes used to deploy Storm and run the application prototype generator NAMB.



4.3 Deployment of Storm in the Cloud Only and the streaming of the prototype generator application NAMB over Storm

As I do with the Sophia + Rennes multi-cluster, I will only do so with the Rennes "Parapluie" cluster which represents my Cloud. I will connect my local laptop to the cluster of Rennes and I will reserve 4 nodes in Parapluie like the two pictures below show:

```
ahafid@access-north:~$ ssh rennes
Linux frennes 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64
----- Grid'5000 - Rennes - frennes.rennes.grid5000.fr -----

This site has 5 clusters (see: https://www.grid5000.fr/w/Rennes:Hardware)

Available in queue default:
 - parapide  (2010): 20 nodes (2 CPUs Intel Xeon X5570, 4 cores/CPU, 24GB RAM, 465GB HDD, 1 x 1Gb Ethernet, 1 x 20Gb InfiniBand)
 - parapluie (2010): 18 nodes (2 CPUs AMD Opteron 6164 HE, 12 cores/CPU, 48GB RAM, 232GB HDD, 1 x 1Gb Ethernet, 1 x 20Gb InfiniBand)
 - paranoia  (2014): 8 nodes (2 CPUs Intel Xeon E5-2660 v2, 10 cores/CPU, 128GB RAM, 5x558GB HDD, 2 x 10Gb Ethernet, 1 x 1Gb Ethernet)
 - parasilo  (2015): 28 nodes (2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 128GB RAM, 5x558GB HDD, 186GB SSD, 2 x 10Gb Ethernet)
 - paravance (2015): 72 nodes (2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 128GB RAM, 2x558GB HDD, 2 x 10Gb Ethernet)

** Useful links:
 - account management (password change): https://api.grid5000.fr/ui/account
 - homepage: https://www.grid5000.fr/mediawiki/index.php/Category:Portal:User
 - charter : https://www.grid5000.fr/mediawiki/index.php/Grid5000:UserCharter
 - support : https://www.grid5000.fr/mediawiki/index.php/Support

** Others sites:
$ ssh {grenoble,lille,luxembourg,lyon,nancy,nantes,sophia}
Last login: Tue Jan 14 15:54:52 2020 from 192.168.66.33
```

```
ahafid@frennes:~$ oarsub -t deploy -p "cluster='parapluie'" -I -l nodes=4,walltime=2
[ADMISSION RULE] Modify resource description with type constraints
[ADMISSION_RULE] Resources properties : \{'property' => 'type = \'default\'','resources' => [{'resource
[ADMISSION RULE] Job properties : ((cluster='parapluie') AND deploy = 'YES') AND maintenance = 'NO'
Generate a job key...
OAR_JOB_ID=1243846
Interactive mode: waiting...
[2020-01-21 13:33:35] Start prediction: 2020-01-21 13:33:35 (FIFO scheduling OK)


[2020-01-21 13:34:40] Start prediction: 2020-01-21 13:34:40 (FIFO scheduling OK)
[2020-01-21 13:35:45] Start prediction: 2020-01-21 13:35:45 (FIFO scheduling OK)
[2020-01-21 13:35:54] Start prediction: 2020-01-21 13:35:54 (FIFO scheduling OK)
[2020-01-21 13:36:12] Start prediction: 2020-01-21 13:36:12 (FIFO scheduling OK)
Starting...
Connect to OAR job 1243846 via the node frontend
```

I will configure the cluster Parapluie in Rennes to support Storm by adapting the file of configuration cluster.conf , I changed the user name to "ahafid" , the deploy image name to be "debian9-storm-base" and I changed specifically the "multi.cluster" to be "no" because I will use only the cluster Parapluie in Rennes.

```
[g5k]
user.name = ahafid
deploy.image.name =debian9-storm-base
# node.memory.mb = 24576.0
node.memory.mb = 32768.0
node.cpu.units = 8
oar.file.location = default
multi.cluster = no

[storm]
storm.version = 1.2.1
zookeeper.version= 3.4.12
csv.log.dir = ../storm_csv_metrics
csv.filter.expression = .*default.*emitted.*
# zookeeper.nodes = 1
nimbus.nodes = 1
workers.per.node = 20
workers.starting.slot = 6700
worker.max.heap.size.mb = 1024.0
worker.heap.memory.mb = 1024
storm.scheduler = org.apache.storm.scheduler.DefaultScheduler
# storm.scheduler = org.apache.storm.scheduler.resource.ResourceAwareScheduler

[ansible]
inventory.file.path = ./g5khosts
playbook.file.path = ./storm_playbook.yaml
~
```

After that I will install Storm in this nodes that I reserved in Parapluie

```
ahafid@frennes:~/g5k-storm-cluster-master$ ls
cluster.conf  debian9-storm-base.env  debian9-storm-base.tgz  deploy.py  g5khosts  LICENSE  README.md  storm_nimbus.yaml  storm_playbook.yaml  st
ahafid@frennes:~/g5k-storm-cluster-master$ python3 deploy.py
```

```
TASK [copy storm supervisor configuratiion file] ***********************************************************************************
[WARNING]: Platform linux on host parapluie-15.rennes.grid5000.fr is using the discovered Python interpreter at /usr/bin/python, but future i
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more infor

changed: [parapluie-15.rennes.grid5000.fr]
[WARNING]: Platform linux on host parapluie-2.rennes.grid5000.fr is using the discovered Python interpreter at /usr/bin/python, but future in
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more infor

changed: [parapluie-2.rennes.grid5000.fr]
[WARNING]: Platform linux on host parapluie-38.rennes.grid5000.fr is using the discovered Python interpreter at /usr/bin/python, but future i
Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more infor

changed: [parapluie-38.rennes.grid5000.fr]

TASK [starting supervisor] *********************************************************************************************************
changed: [parapluie-15.rennes.grid5000.fr]
changed: [parapluie-2.rennes.grid5000.fr]
changed: [parapluie-38.rennes.grid5000.fr]

PLAY RECAP *************************************************************************************************************************
parapluie-13.rennes.grid5000.fr : ok=5    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-15.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-2.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
parapluie-38.rennes.grid5000.fr : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Waiting startup...
curl 'http://parapluie-13.rennes.grid5000.fr:8080/api/v1/supervisor/summary'
curl: (7) Failed to connect to parapluie-13.rennes.grid5000.fr port 8080: Connection refused


**** ssh tunnel ***
ssh ahafid@access.grid5000.fr -N -L8080:parapluie-13.rennes.grid5000.fr:8080
ahafid@frennes:~/g5k-storm-cluster-master$
```

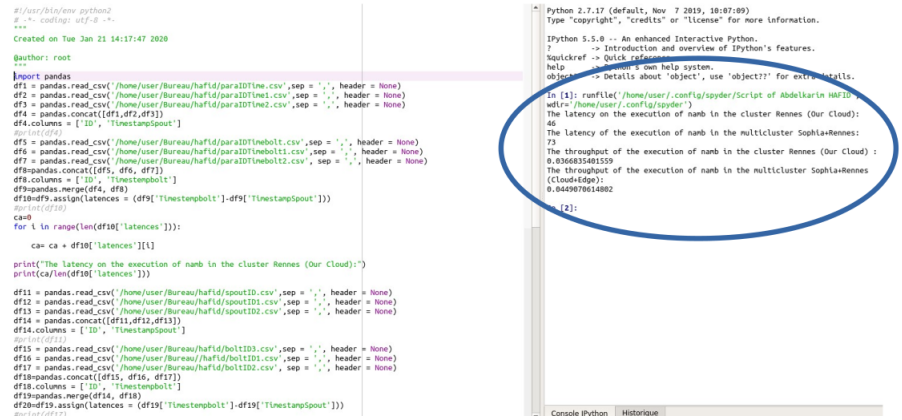
Furthemore, I will run the prototype NAMB over Storm installed in the Cloud Rennes.

```
ahafid@frennes:~/namb-0.7.1$ ls
LICENSE  README.md  benchmarks  conf  modules  namb.py  namb.yml  pom.xml
ahafid@frennes:~/namb-0.7.1$ python3 namb.py storm -p ~/apache-storm-1.2.1/bin/storm
Running: java -client -Ddaemon.name= -Dstorm.options= -Dstorm.home=/home/ahafid/apache-storm-1.2.1 -Dstorm.log.di
ath=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file= -cp /home/ahafid/apache-storm-1.2.1/*:/home/ahafid/
1/extlib/*:/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar:/home/ahafid/apache-storm-
jar=/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar -Dstorm.dependency.jars= -Dstorm.
blication /home/ahafid/namb-0.7.1/conf/namb.yml /home/ahafid/namb-0.7.1/conf/storm-benchmark.yml
spout_1 bolt_1 bolt_2 2421 [main] WARN  o.a.s.u.Utils - STORM-VERSION new 1.2.1 old null
2507 [main] INFO  o.a.s.StormSubmitter - Generated ZooKeeper secret payload for MD5-digest: -6587358101405030065:
2802 [main] INFO  o.a.s.u.NimbusClient - Found leader nimbus : parapluie-13.rennes.grid5000.fr:6627
2861 [main] INFO  o.a.s.s.a.AuthUtils - Got AutoCreds []
2868 [main] INFO  o.a.s.u.NimbusClient - Found leader nimbus : parapluie-13.rennes.grid5000.fr:6627
2934 [main] INFO  o.a.s.StormSubmitter - Uploading dependencies - jars...
2935 [main] INFO  o.a.s.StormSubmitter - Uploading dependencies - artifacts...
2936 [main] INFO  o.a.s.StormSubmitter - Dependency Blob keys - jars : [] / artifacts : []
2955 [main] INFO  o.a.s.StormSubmitter - Uploading topology jar /home/ahafid/namb-0.7.1/benchmarks/storm-bench/ta
apache-storm-1.2.1/local-dir/nimbus/inbox/stormjar-ed341c3e-f146-4121-b2db-ab5615c7b25c.jar
Start uploading file '/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar' to '/root/apac
-f146-4121-b2db-ab5615c7b25c.jar' (7040562 bytes)
[==================================================] 7040562 / 7040562
File '/home/ahafid/namb-0.7.1/benchmarks/storm-bench/target/storm-bench-0.7.1.jar' uploaded to '/root/apache-stor
121-b2db-ab5615c7b25c.jar' (7040562 bytes)
3203 [main] INFO  o.a.s.StormSubmitter - Successfully uploaded topology jar to assigned location: /root/apache-st
-4121-b2db-ab5615c7b25c.jar
3204 [main] INFO  o.a.s.StormSubmitter - Submitting topology namb_bench_1579610893394 in distributed mode with co
topology.auth.scheme":"digest","storm.zookeeper.topology.auth.payload":"-6587358101405030065:-4876267379485461731
3204 [main] WARN  o.a.s.u.Utils - STORM-VERSION new 1.2.1 old 1.2.1
3822 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: namb_bench_1579610893394
```

I retrieved the log files generated in the worker nodes in the Cloud: Parapluie and the log Files generated in FoG+Cloud: Suno+Parapluie to parse them and get the informations that I will use to calculate the latency and the throughput of the execution of NAMB over Storm in the two cases. The log files after parsing them looks like the image below:

[bolt_1_7]	2798c394-4e7a-4ec5-a5cc-ff3e24063822	400	1579014662312
[bolt_3_13]	cfdeca36-3fbc-41c2-be82-02763af4d56c	1000	1579014662317
[bolt_3_13]	e89c3b0f-1b71-4b8c-af37-ba69c933fbaf	1200	1579014662319
[bolt_3_13]	76653144-94c5-44fb-bb08-02c7ea6ce6ae	1400	1579014662322
[bolt_3_13]	d3e55dad-2946-4af9-a129-7f099952a21c	1600	1579014662326
[bolt_3_13]	42e97dc6-929f-4bef-a24d-524b64e0bc06	1800	1579014662330
[bolt_3_13]	91e5f393-6155-4e68-a915-3f4f83721551	2000	1579014662333
[bolt_1_7]	2011ed13-efce-46e9-b763-cce06a8ba67b	600	1579014662334
[bolt_3_13]	a3bbc315-bf7c-4e12-8ed0-c0543bb35690	2200	1579014662337
[bolt_3_13]	d7d60110-1942-4721-9e59-1f545d823adc	2400	1579014662341
[bolt_1_7]	4c6baaf0-0d93-40d0-8367-8303439e3494	800	1579014662346
[bolt_1_7]	4bed5f92-7782-48ff-978d-f0cde583a7b9	1000	1579014662356
[bolt_3_13]	cd939fe4-d0c7-4a1f-bb01-0b247deb4760	2600	1579014662367
[bolt_1_7]	fa8be2c6-9c11-4b1e-bdcf-7635a9105a7f	1200	1579014662376
[bolt_1_7]	2dc6f4e8-52ca-4519-a443-d2eb2355e6b5	1400	1579014662389
[bolt_3_13]	e83fbd11-5e41-46ad-b3f4-5cbe5f2e38ee	2800	1579014662396
[bolt_1_7]	e7e09124-a391-464f-b843-8b96a468b466	1600	1579014662396
[bolt_1_7]	267b45d8-3dd9-4d15-9f05-985056de94f4	1800	1579014662404
[bolt_3_13]	e75903e3-f363-47f4-830e-ba666f0048ad	3000	1579014662415
[bolt_1_7]	299ffd80-8edc-45d0-9cf5-762178dcf99c	2000	1579014662422
[bolt_1_7]	150e01b5-ba38-43e0-951d-b46cc6bc6fc6	2200	1579014662429
[bolt_1_7]	2dd56159-5aa0-49f4-b426-588a7deddcc0	2400	1579014662435
[bolt_3_13]	81151031-cc47-45ad-8fb2-400e4545adf3	3200	1579014662440
[bolt_1_7]	551c40b5-5c70-4077-87bd-c2a99129e9b9	2600	1579014662443
[bolt_1_7]	267c86ee-8172-49a7-af98-5c06dbbdf983	2800	1579014662451
[bolt_1_7]	8020e706-5c96-4739-a4d8-b588e8c2a50c	3000	1579014662457
[bolt_3_13]	ffe08967-068d-4a7f-936f-bacdc1c17d55	3400	1579014662463
[bolt_1_7]	5d5a3ef6-eb35-4717-bcdd-797f59458154	3200	1579014662463
[bolt_1_7]	eff5ebc4-2dac-4cc6-a94b-7a1bc7b2203a	3400	1579014662470
[bolt_1_7]	364f524f-6900-4ce8-88f7-5fe76dda53b9	3600	1579014662481

In fact, I used the Python Pandas library to manipulate all the log files I previously retrieved as data frames and to have the ability to concatenate and merge these data frames to calculate the throughput and latency of NAMB execution in the first time in the FoG+cloud: Sophia+Rennes environment and secondly in the Cloud only in Rennes. My goal from these experiences is to prove that the throughput and the latency of NAMB execution in a FoG+Cloud environment (Sophia + Rennes) are higher than the throughput and latency of NAMB execution in the cloud only (Rennes). For the full script, you will find it zipped with this report



4.4 The Final results of our experimentation's

The results of the execution of the script python that I developed are encircled in Blue in the image above and they are Clair in this picture:

```
In [1]: runfile('/home/user/.config/spyder/Script of Abdelkarim HAFID', wdir='/home/user/.config/spyder')
The latency of the execution of the prototype generator NAMB in the Cloud only : the cluster Parapluie in Rennes
46
The latency of the execution of the protype generator NAMB in the FOG+Cloud : the multicluster Suno+Parapluie in Sophia+Rennes:
73
The throughput of the execution of the prototype generator NAMB in the Cloud only : the Cluster Parapluie in Rennes
0.0366835401559
The throughput of the execution of the prototype NAMB in the FOG+Cloud : the multicluster Suno+Parapluie in Sophia+Rennes
0.0449070614802
```

This results show that the latency of the execution of the prototype generator NAMB in the cloud only is 46ms and the latency of the execution of the prototype generator

NAMB in the Fog+Cloud: Sophia+Rennes is 74ms.So,I proved one drawback of the Fog+Cloud architecture is that the latency is high in the Fog+Cloud architecture than in the Cloud one .However, the experimentation's shows that the throughput in the Fog+Cloud is: 45 tuples /s, so it is high that the throughput in the Cloud only: 36 tuples/s the fact which proves the crucial advantage of the environment Fog+Cloud, we are therefore faced with a compromise between throughput and latency in the two different architectures.

Chapter 5

Conclusion

In this PFE, we have tried to study and define the advantages and the drawbacks of data streaming in Fog environments. The ineffectiveness of data streaming in the cloud only led us to look for use cases where the Fog + Cloud architecture took place: in this report, you find 4 use cases that use the Fog computing solution to improve the performance of application's streaming . In addition to these use cases, Big Data Stream processing on the Fog+Cloud environment has generated a new wave of research trying to understand and identify the impact of various parameters on the performance of applications executed on the Fog+Cloud environment, aiming to find the most effective strategy to deploy this architecture. For us, we have proven by one experimentation that the throughput in the FoG+Cloud environment is higher than the throughput in the Cloud only. So, one crucial advantage of the Fog+Cloud architecture takes place. However, we have to pay the latency: because we found that the latency of the streaming of the prototype NAMB in the Cloud is lower than the latency in the Fog+Cloud. In the context of one future work, we can try to study how to establish a compromise between the gain of the throughput and the loss of the latency in the FOG+Cloud architecture.

Bibliography

[1]:Abdulaziz Alarifi 1 , Fathi Abdelsamie 2 , Mohammed Amoon A fault-tolerant aware scheduling method for fog-cloud environments

[2]:B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in IEEE Smart-Cloud, 2016

[3]: Tarek R.Sheltami,Essa Q.Sahra,Elhadi M.Shakshuki. "Fog computing : Data Streaming Services for mobile end users".

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

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

[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] PrEstoCloud: Proactive Cloud Resources Management at the Edge for Efficient Real-Time Big Data Processing.

[8] PrEstoCloud GA 73339 Deliverable 7.1, the Editor of this deliverable is : Noam Amram and the submission date is 05/04/2018.

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

[10]B. Brehmer, "The dynamic ooda loop : Amalgamating boyd s ooda loop and the cybernetic approach to command and control assessment , tools and metrics," 2005.

[11]A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: A real-time iot benchmark for distributed stream processing platforms," CoRR, vol. abs/1701.08530, 2017.

[12]Eduard Gibert Renart , Alexandre da Silva Veith † , Distributed Operator Placement for IoT Data Analytics Across Edge and Cloud Resources