ORCHESTRATION OF DISTRIBUTED INGESTION AND PROCESSING OF IOT DATA FOR FOG PLATFORMS

JUAN LUIS PÉREZ

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor in Computer Science

Universitat Politècnica de Catalunya

2018

Juan Luis Pérez: *Orchestration of Distributed Ingestion and Processing of IoT Data for Fog Platforms,* A dissertation submitted in partial fulfillment of the requirements for the degree of "Doctor per la Universitat Politècnica de Catalunya".

ADVISOR:
David Carrera

AFFILIATION:
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya

LOCATION:
Barcelona

ABSTRACT

In recent years there has been an extraordinary growth of the Internet of Things (IoT) and its protocols in both, industry and academia. The increasing diffusion of electronic devices with identification, computing and communication capabilities, such as mobile phones, digital cameras, and music players, is laying ground for the emergence of a highly distributed service and networking environment. The rapid evolution of the mobile Internet, embedded distributed devices and Machine-to-Machine (M2M) communication in the Cloud has enabled the IoT technologies.

From the point of view of predictions, Gartner expects that IoT will grow to nearly 21 billion connected things by 2020. Companies like Cisco and Ericsson predict that it will hit 50 billion by 2020. Regardless of the dance of numbers, what is undeniable is the importance of the Internet of Things for both the industry and the academy now as in the coming years. IoT hardware will be the largest technology category in 2018 with $239 billion going largely toward modules and sensors along with spending on infrastructure and security.

The above mentioned situation implies that there is an increasing demand for advanced IoT data management and processing platforms. Such platforms require support for multiple protocols at the edge for extended connectivity with the objects, but also need to exhibit uniform internal data organization and advanced data processing capabilities to fulfill the demands of the application and services that consume IoT data.

One of the initial approaches to address this demand is the integration between IoT and the Cloud computing paradigm. There are many benefits of integrating IoT with the Cloud computing. The IoT generates massive amounts of data, and Cloud computing provides a pathway for that data to travel to its destination. But today's Cloud computing models do not quite fit for the volume, variety, and velocity of data that the IoT generates.

Among the new technologies emerging around the Internet of Things (IoT) to provide a new whole scenario, the Fog Computing paradigm has become the most relevant. Fog computing was introduced a few years ago in response to challenges posed by many IoT applications, including requirements such as very low latency, real-time operation, large geo-distribution, and mobility. Also this low latency, geo-distributed and mobility environments are covered by the network architecture MEC (Mobile Edge Computing) that provides an IT service environment and Cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network (RAN) and in

close proximity to mobile subscribers. Fog computing addresses use cases with requirements far beyond Cloud-only solution capabilities. The interplay between Cloud and Fog computing is crucial for the evolution of the so-called Internet of Things (IoT), but the reach and specification of such interplay is an open problem.

This thesis aims to find the right techniques and design decisions to build a scalable distributed system for the IoT under the Fog Computing paradigm to ingest and process data. The final goal is to explore the trade-offs and challenges in the design of a solution from Edge to Cloud to address opportunities that current and future technologies will bring in an integrated way. This thesis describes an architectural approach that addresses some of the technical challenges behind the convergence between IoT, Cloud and Fog with special focus on bridging the gap between Cloud and Fog. To that end, new models and techniques are introduced in order to explore solutions for IoT environments. More specifically, it is focused in three scenarios that are incremental in its scope. First, it studies an architectural approach under the Cloud paradigm to manage services under an IoT environment. Second, it examines the feasibility and benefits of a Fog computing IoT solution under a smart city scenario. And finally, it brings a final architectural proposal for the orchestration of ingestion and processing IoT data for a Fog platform.

Following these three items described above, this thesis contributes to the architectural proposals for IoT ingestion and data processing by 1) proposing the characterization of a platform for hosting IoT workloads in the Cloud providing multi-tenant data stream processing capabilities, the interfaces over an advanced data-centric technology, including the building of a state-of-the-art infrastructure to evaluate the performance and to validate the proposed solution. 2) studying an architectural approach following the Fog paradigm that addresses some of the technical challenges found in the first contribution. The idea is to study an extension of the model that addresses some of the central challenges behind the converge of Fog and IoT. 3) Design a distributed and scalable platform to perform IoT operations in a moving data environment. The idea after study data processing in Cloud, and after study the convenience of the Fog paradigm to solve the IoT close to the Edge challenges, is to define the protocols, the interfaces and the data management to solve the ingestion and processing of data in a distributed and orchestrated manner for the Fog Computing paradigm for IoT in a moving data environment.

ACKNOWLEDGMENTS

This work is partially supported by the Ministry of Science and Technology of Spain under contracts TIN2012-34557, 2014SGR1051 and TIN2015-65316-P, by the BSC-CNS Severo Ochoa program (SEV-2011-00067 and SEV-2015-0493) and Generalitat de Catalunya under contract 2014SGR1051, by the ICREA Academia program, and by the by the European Commission IST activity of the 7th Framework Program under contract number 317862 (COMPOSE). It is also by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 639595). And it is also partially supported by Cisco Systems, Inc.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

1.1 MOTIVATION

Currently we live in a hyperconnected world where any device is likely to be connected, from a car to a clock, passing through a washing machine or a refrigerator. Any *"thing"* in the real world has the ability to be connected through embedded solutions. All these things have the ability to communicate with each other. This network of physical devices, vehicles, home appliances and other items embedded with electronics, software and sensors which enables these things to connect and exchange data is the Internet of Things (IoT). Thanks to the high growth of mobile networks, Machine-to-Machine [67] (M2M) communication protocols, mini-hardware manufacturing, micro-computing and cheap processors the IoT technologies have been enabled.

Gartner expects that IoT will grow to nearly 21 billion connected things by 2020 [14]. Cisco [3] and Ericsson [11], on the other hand, predict that it will hit 50 billion by 2020. Regardless of the dance of numbers, what is undeniable is the importance of the Internet of Things for both the industry and the academy now as in the coming years. IoT hardware will be the largest technology category in 2018 with $239 billion going largely toward modules and sensors along with spending on infrastructure and security [15].

More business are becoming aware of the relevance of the IoT and the amount of data IoT is able to gather, but it is not just a problem of volume of data, IoT raises problems far beyond data volume. The Internet of Things components basically cover a variety of the latest technologies used in the present era and require solving a variety of research questions at different architecture layers. The number of

different players in the market covers a wide range, both horizontally, in terms of functionality, and vertically, among different industries. Another issue is that the full benefits of the Internet of Things are realized when large enough number of devices are able to interact with each other, and therein lies a big problem.

The core infrastructure of an IoT environment is formed by sensors, actuators, compute servers, and the communication network, and the application layer provides the framework for communication and distributed application development that includes data mining, data processing and visualization APIs. The IoT environment might support the interaction between things and allow for paradigms like distributed computing. Internet of Things frameworks might lead to software-development environments to help the development of software to work with IoT components

At the same time, connected things need to be able to speak to each other to transfer data and therefore a lot of standardization to connect many different devices is needed. Despite the availability of open standards and the standards organization that aspire to successfully setting their own standards, there is still room for improvement, specially with regards to the integration of the different standards between them.

Based on the scenario described above, Internet of Things is not only a Big Data problem, a networking problem or a communication problem, can be defined as a multidisciplinary problem where the different topics need to be address in order to propose an overall solution.

Moreover, in recent years there has been an extraordinary growth of large-scale data processing technologies in both, industry and academic communities. This growth is due by the need to process the enormous amount of data that both companies and communities must be able to handle, and has led the introduction of new architectures and infrastructures. Besides that, the set up of Cloud computing as the paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services have also contributed significantly to the growth of large-scale technologies. Cloud computing has completely changed the way servers, storage, and other IT resources are delivered, posing new challenges to data centers. Therefore, it is normal that the first solutions for the Internet of Things came from the Cloud Computing, but the volume, variety, and velocity of data that IoT generates and all the particular considerations for the sensors and actuators included in IoT frameworks make it necessary a specific approach to the solution.

One example of the kind of emerging technologies that is worth targeting is the Fog Computing paradigm. The Fog Computing paradigm allows to bring the Cloud Computing to the Edge. Both Cloud Computing and Fog Computing provide storage, applications, and data

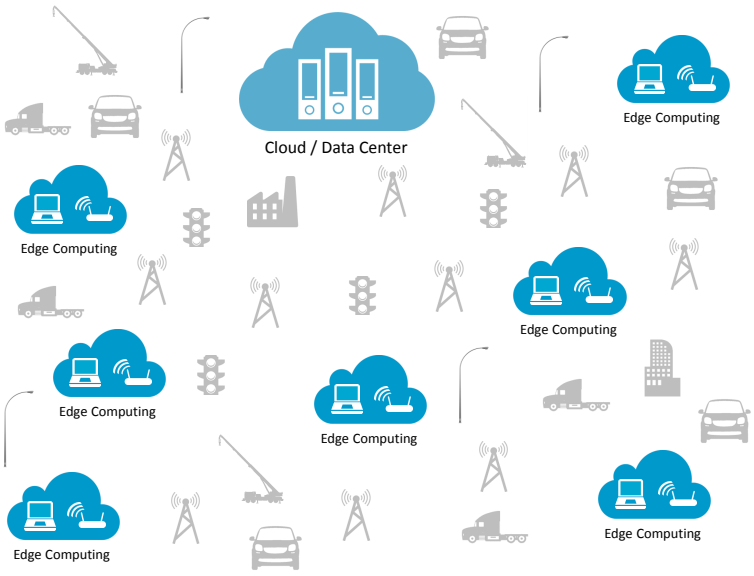


Figure 1.1: IoT Ecosystem

to end-users. However, Fog Computing has a bigger proximity to end devices and bigger geographical distribution. Fog Computing facilitates the operation of compute, storage, and networking services between end devices and Cloud Computing data centers. The missing link for what data needs to be pushed to the Cloud, and what can be analyzed locally, at the Edge, is provided by Fog Computing. From the point of view of the Internet of Thing, Fog Computing is a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Edge.

Fog Computing addresses some of the problems discussed above. Fog Computing can create low-latency network connections between devices and analytics endpoints. This architecture in turn reduces the amount of bandwidth needed compared to if that data had to be sent all the way back to Cloud or a data center for processing. Fog Computing also provides an approach where the communication between end devices is inherent to its architecture. Fog environments should be horizontally scalable supporting multiple industry vertical use cases. Fog environments has to be able to work across the Cloud to Things. And be a system-level technology, that extends from Things, over network Edges, through to the Cloud and across various network protocols, and fitting with the advances in 5G [1] Radio Access Network (RAN) and Mobile Edge Computing (MEC) architecture that are also key for the IoT evolution.

The complementarity between Fog and Cloud has traditionally been seen as a mandatory feature in any Fog platform. This thesis advocates for a different approach. Rather than specifying an architecture where Fog and Cloud are complementary by design, is focused on a

service management architecture that literally fuses Fog and Cloud. The final goal of this work is to explore the trade-offs and challenges in the design of a solution from Edge to Cloud to address opportunities that current and future technologies will bring in an integrated way. Figure 1.1 represents the IoT ecosystem diversity.

This thesis aims to study and address these problems with the idea of improving the integration of Cloud with Fog in an IoT environment, allowing alternative architecture solutions for orchestration of distributed ingestion and processing of IoT data. More specifically, it is focused in three areas. First, it will explore an architectural approach under the Cloud paradigm to manage services under an IoT environment proposing the characterization of a platform for hosting IoT workloads in the Cloud providing mutli-tenant data stream processing capabilities, the interfaces over an advanced data-centric technology, including the building of a state-of-the-art infrastructure to evaluate the performance and to validate the proposed solution.. Second, it will explore the feasibility and benefits of a Fog computing IoT solution under a smart city scenario studying an architectural approach following the Fog paradigm that addresses some of the technical challenges found in the first contribution. The idea is to study an extension of the model that addresses some of the central challenges behind the converge of Fog and IoT. And finally, it brings a final architectural proposal for the orchestration of ingestion and processing IoT data for a Fog platform designing a distributed and scalable platform to perform IoT operations in a moving data environment. The idea after study data processing in Cloud, and after study the convenience of the Fog paradigm to solve the IoT close to the Edge challenges, is to define the protocols, the interfaces and the data management to solve the ingestion and processing of data in a distributed and orchestrated manner for the Fog Computing paradigm for IoT in a moving data environment.

1.2 CONTRIBUTIONS

The contributions of this thesis revolve around the interfaces, the orchestration and the architectural solutions to the problem of data ingestion in distributed environments. All the work done is incremental in that each contribution is based on the previous one, but at the same time each one of them delves into a new topic and proposes solutions to different problems.

Figure 1.2 illustrates the two main directions this thesis explores, and the steps taken in each one of them. The first direction represents the Scope of this thesis and the second direction represents the Value that this thesis try to achieve. In this figure, the crossing between each one of the scopes and its value represents each one of the

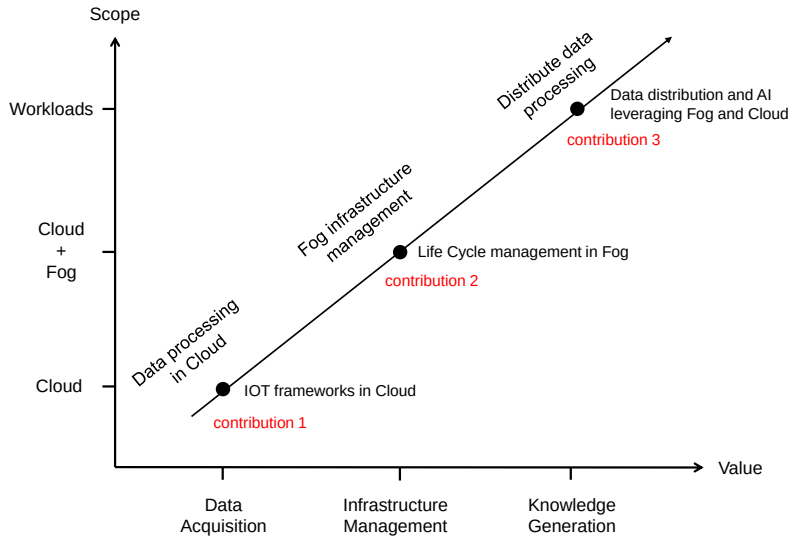


Figure 1.2: Major steps for each contribution

contributions of this thesis. More details about each contribution are provided in the following sections.

This work aims to demonstrate the feasibility of the following thesis:

It is possible to develop an unified and distributed orchestration layer for data ingestion and processing based on the Fog Computing paradigm for IoT in a moving data sources environment.

To reach this statement, three are the major research objectives that are addressed in this thesis. The first goal is develop interfaces for data processing oriented to the IoT multi-tenancy and sharing data between tenants, the second is develop orchestration techniques and interfaces to manage life cycle of data processing elements in a Fog environment based on the Fog Computing paradigm and the third is develop a distributed and scalable platform to perform IoT operations in a environment with moving data sources. More details about each contribution are provided in the following sections.

1.2.1 *Hosting IoT data-centric workloads in the Cloud*

Over the last years, Internet of Things (IoT) and Big Data platforms are clearly converging in terms of technologies, problems and approaches. IoT ecosystems generate a vast amount of data that needs to be stored and processed, becoming a Big Data problem. IoT devices and sensors generate streams of data across a diversity of locations and protocols that in the end reach a central platform that is used to store and process it. Processing can be done in real time, with transfor-

mations and enrichment happening on-the-fly, but it can also happen after data is stored and organized in repositories. In the former case, realtime processing technologies are required to operate on the data; in the latter analytics and queries are of common use.

The **first contribution** of this thesis is the characterization of a platform for hosting IoT workloads in the Cloud providing multi-tenant data stream processing capabilities, a REST API over an advanced data-centric technologies. The work includes the building of a state-of-the-art infrastructure to evaluate the performance and to validate the viability of the proposed solutions.

The work performed in this area has resulted in the following main publications:

[65] Juan Luis Pérez, Álvaro Villalba, David Carrera, Iker Larizgoitia, and Vlad Trifa. The COMPOSE API for the internet of things. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 971–976. ACM, 2014. ISBN 978-1-4503-2745-9. doi: 10.1145/2567948.2579226. URL http://doi.acm.org/10.1145/2567948.2579226

[64] Juan Luis Pérez and David Carrera. Performance characterization of the servioticy API: an iot-as-a-service data management platform. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 62–71. IEEE Computer Society, 2015. ISBN 978-1-4799-8128-1. doi: 10.1109/BigDataService.2015.58. URL https://doi.org/10.1109/BigDataService.2015.58

[80] Álvaro Villalba, Juan Luis Pérez, David Carrera, Carlos Pedrinaci, and Luca Panziera. servioticy and iserve: A scalable platform for mining the iot. In Elhadi M. Shakshuki, editor, *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), London, UK, June 2-5, 2015*, volume 52 of *Procedia Computer Science*, pages 1022–1027. Elsevier, 2015. doi: 10.1016/j.procs.2015.05.097. URL https://doi.org/10.1016/j.procs.2015.05.097

1.2.2 *Distribution of data processing under the Fog paradigm*

The interplay between Cloud and Fog computing is crucial for the evolution of IoT, but the reach and specification of such interplay is an open problem. Meanwhile, the advances made in managing hyper-distributed infrastructures involving the Cloud and the network Edge are leading to the convergence of NFV and 5G, supported mainly by ETSI's MANO architecture.

The **second contribution** of this thesis is a study of an architectural approach following the Fog paradigm that addresses some of the technical challenges found in the first contribution. The idea is to study an extension of the model that addresses some of the central challenges behind the convergence of Fog and IoT.

This second contribution has been done in collaboration with the Corporate Strategic Innovation Group (CSIG) from Cisco and other companies as well as the municipality of Barcelona. Through this collaboration, the candidate participated in the development of a Proof-of-Concept (PoC) on Fog Computing that was deployed in the streets of Barcelona, involving the provisioning, deploying, managing, and maintaining the compute, network, and storage resources needed for running Fog Services. The contributions from the candidate to this international collaboration relate to the solution architecture design, data processing techniques, life cycle management and API interfaces extension, mainly:

- Managing the lifecycle of the Fog Nodes.
- Managing the lifecycle of the Fog Virtual Domains (FVDs), i.e., the virtual environments supporting the different Applications running within a Fog Node.
- The APIs and protocols required for managing a Fog System, including the interfaces and protocols: i) within the Backend Platform; ii) within the Fog Nodes; iii) between the Backend Platform and the Fog Nodes; iv) and between the Backend Platform and the External Management Systems.

The work performed in this area has resulted in the following main publications:

[86] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, and Albert Olive. A new era for cities with fog computing. *IEEE Internet Computing*, 21 (2):54–67, 2017. doi: 10.1109/MIC.2017.25. URL https://doi.org/10.1109/MIC.2017.25

[79] Frank van Lingen, Marcelo Yannuzzi, Anuj Jain, Rik Irons-Mclean, Oriol Lluch Parellada, David Carrera, Juan Luis Pérez, Alberto Gutierrez, Diego Montero, Josep Marti, Ricard Maso, and Juan Pedro Rodriguez. The unavoidable convergence of nfv, 5g, and fog: A model-driven approach to bridge cloud and edge. *IEEE Communications Magazine*, 55(8):28–35, 2017

1.2.3 *Distribution and scalability of IoT operations with moving data sources*

After the first two contributions where we explore the trade-offs and challenges in the design of architectures for IoT based in Cloud and Fog paradigms we presented a converged Cloud/Fog architecture that addresses the challenges currently IoT is facing and an application running inside.

The **third contribution** of this thesis is a proposed architecture for a city-wide traffic service based on the Fog Computing paradigm. We propose a data distribution algorithm resilient to back-haul connectivity issues.

The work performed in this area has resulted in the following publications:

[66] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll. Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. *Future Generation Computer Systems*, 87:198 – 212, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.05.013. URL http://www.sciencedirect.com/science/article/pii/S0167739X1732678X

1.3 THESIS ORGANIZATION

The remaining chapters of this thesis are organized as follows. Chapter 2 introduces some basic concepts related to Internet of Things and Web of Things, Cloud and Fog computing and Edge oriented architectures. Chapter 3 introduces the building of a state-of-the-art infrastructure for hosting IoT workloads in the Cloud and its characterization to evaluate the performance. Chapter 4 presents an architecture that addresses some of the central challenges behind the convergence of NFV, 5G/MEC, IoT, and Fog. Chapter 5 is focused on presenting a decentralized architecture towards smart-cities traffic monitoring and forecasting. And finally, Chapter 6 presents the conclusions and future work of this thesis.

2

BACKGROUND

2.1 IOT AND WEB OF THINGS

The Internet of Things may be a hot topic in the industry but it's not a new concept. In the early 2000's, Kevin Ashton was laying the groundwork for what would become the Internet of Things (IoT) at MIT's AutoID lab [32].

Actually, the Internet of Things (IoT) is an integrated part of the Future Internet and could be defined as a dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.

In the IoT, "things" are expected to become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information "sensed" about the environment, while reacting autonomously to the "real/physical world" events and influencing it by running processes that trigger actions and create services with or without direct human intervention.

The technology of IoT has been evolved according to the environment based on information communication technology and social infrastructure. By connecting billions or even trillions of devices to the Internet, we realize that there are a lot of applications that are being used by the industries, the government, the public, etc. Massive amounts of data are being generated by billions of connected devices and transferred throughout the network to the Internet.

In the IoT topic there must be an integration between the physical world and computer networks. The things have to be allowed based on competing standards and requires custom solutions, hence requires time and technical expertise. In this heterogeneous ecosystem of devices, the development of a simple application requires knowledge and time. Real world devices can be integrated to the Web. REST principles can be applied to devices.

The Web of Things proposes to leverage the existing and ubiquitous Web protocols as common ground where real objects could interact with each other. One of the advantages of using Web standards is that devices will be able to finally "speak" the same language as other resources on the Internet, therefore making it very easy to integrate physical devices with any content on the Web.

The Web of Things is designed to be seamlessly integrated to the existing Web so it can fully leverage its infrastructure and standards to minimize integrations across applications and systems. The definition of the fundamental building blocks of the Web of Things is an extension of the current Web paradigms.

Web of Things proposes to integrate real world things int the existing Web by turning real objects into RESTful resources that can be used directly over HTTP. A Web Server can be implemented on embedded devices to turn them into a RESTful resource.

The Web Thing Model Submission is a specification that was published in 2015 [81] by several members of the WoT IG. This document proposes the basis of a common model to describe the virtual counterpart of physical objects in the Web of Things. It defines a model and Web API for Things to be followed by anyone wanting to create a product, device, service, or application for the Web of Things.

2.2 CLOUD COMPUTING

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [61]. It is a computing technique that allows renting storage infrastructures and computing services, renting of business processes and overall applications. It simplifies the computing jobs by renting resources and services.

There are many benefits of integrating IoT with the Cloud. The IoT generates massive amounts of data, and cloud computing provides a pathway for that data to travel to its destination. IoT is a network of devices which act as information sources and produces a colossal amounts of semi-structured or non-structured data with the three typical Big Data characteristics: volume, velocity and variety. Into the

Cloud, data can be treated in homogeneous manner through standard APIs.

From the point of view of the computational resources, IoT devices possess quite limited processing capacity which do not permit on-site data processing. The collected data from these devices is usually transmitted to the other powerful nodes capable of aggregation and processing.

Cloud systems are located within the Internet, which is a large heterogeneous network with numerous speeds, technologies and topologies without central control. Because of the nature of the Internet, there are many issues related to the quality of service that remain unresolved. One of the most important that affects the quality of service is network latency. Realtime applications are affected by the delay caused by latency in networks.

Today's Cloud computing models are not designed for the volume, variety, and velocity of data that the IoT generates. Billions of previously unconnected devices are generating more than two exabytes of data each day. An estimated 50 billion "things" will be connected to the Internet by 2020 [77]. Moving all data from these things to the cloud for analysis would require vast amounts of bandwidth.

Cloud computing frees the enterprise and the end user from the specification of many details. This facility becomes a problem for latency sensitive applications that require large numbers of nodes in order to meet the delay requirements. An emerging wave of Internet deployments requires mobility support and wide range of Geo-distribution in addition to location awareness and low latency features.

2.3 FOG COMPUTING

An emerging wave of Internet deployments, most notably the Internet of Things (IoTs), requires mobility support and geo-distribution in addition to location awareness and low latency. Analyzing IoT data close to where it is collected minimizes latency. It offloads gigabytes of network traffic from the core network. And it keeps sensitive data inside the network. A new platform is needed to meet these requirements.

Fog computing is a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things.

Fog computing is an extension of the Cloud computing paradigm to the edge of the network where implementations of the architecture can reside in multiple layers of a network's topology.

All the benefits of Cloud should be preserved with the Fog extension, including containerization, virtualization, orchestration, man-

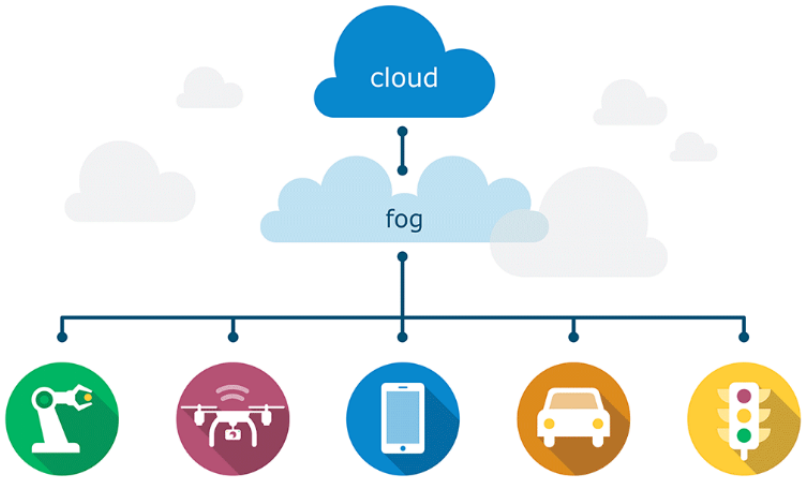


Figure 2.1: Fog computing layer architecture.

ageability, and efficiency. It is important to note that Fog computing complements, not replaces, Cloud computing.

Fog computing supports multiple industry verticals and application domains, delivering intelligence and services to users and business. It enables services and applications to be distributed closer to Things, and anywhere along the continuum between Cloud and Things. It extends from the Things, over the network edges, through the Cloud, and across multiple protocol layers systems. A general view of the layer architecture of Fog computing can be seen in Figure 2.1.

The Fog computing model moves computation from the cloud closer the edge, and potentially right up to the IoT sensors and actuators. The computational, networking, storage and acceleration elements of this new model are known as fog nodes. These are not completely fixed to the physical edge, but should be seen as fluid system of connectivity.

Fog computing targets cross-cutting concerns like the control of performance, latency and efficiency are also key to the success of fog networks. Certain functions are naturally more advantageous to carry out in fog nodes, while others are better suited to cloud. The segmentation of what tasks go to fog and what goes to the backend cloud are application specific.

The OpenFog Reference Architecture enables fog-cloud and fog-fog interfaces. OpenFog architectures offer several unique advantages over other approaches, which we term SCALE:

- **S**ecurity: Additional security to ensure safe, trusted transactions.
- **C**ognition: awareness of client-centric objectives to enable autonomy.

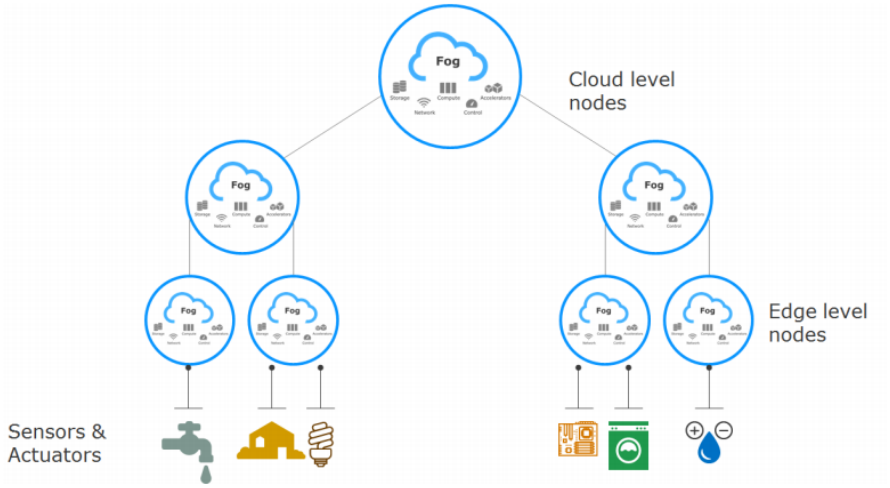


Figure 2.2: Multi-Tier Fog Deployment

- **A**gility: rapid innovation and affordable scaling under a common infrastructure.
- **L**atency: real-time processing and cyber-physical system control.
- **E**fficiency: dynamic pooling of local unused resources from participating end-user devices.

In a typical Fog deployment, there are usually several tiers (N-tiers) of nodes. Nodes at the edge are typically focused on sensor data acquisition, data normalization and command/control of sensors actuators. Nodes in the next higher tier are focused on data filtering, compression, and transformation. They may also provide some edge analytics required for critical real time or near real time processing. As we move away from the true network edge, we see higher level machine and system learning (analytics) capabilities. Nodes at the higher tiers or nearest the backend cloud are typically focused on aggregating data and turning the data into knowledge. Figure 2.2 shows this typical multi-tier fog deployment.

The OpenFog Reference Architecture 2.3 description is a composite of perspectives and multiple stakeholder views. The abstract architecture includes perspectives, shown in grey vertical bars on the sides of the architectural description. The perspectives include:

- Performance: Low latency is one of the driving reasons to adopt Fog architectures. There are multiple requirements and design considerations across multiple stakeholders to ensure this is satisfied. This includes time critical computing, time sensitive networking, network time protocols, etc.
- Security: End-to-end security is critical to the success of all Fog computing deployment scenarios.

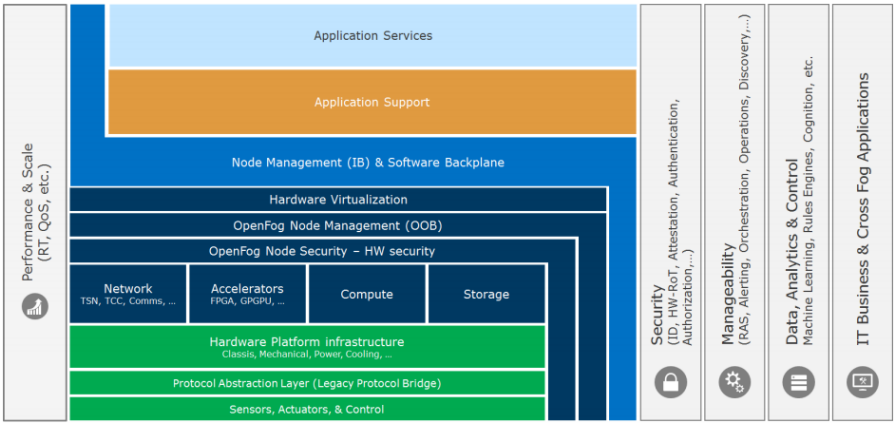


Figure 2.3: OpenFog architecture description with perspectives

- Manageability: Managing all aspects of fog deployments, which include RAS (Reliability, Availability, Serviceability), DevOps, etc., is a critical aspect across all layers of a Fog computing hierarchy.
- Data Analitycs and Control: The ability for Fog nodes to be autonomous requires localized data analytics coupled with control. The actuation/control needs to occur at the correct tier or location in the hierarchy as dictated by the given scenario.
- IT Business and Cross Fog Applications: In a multi-vendor ecosystem applications need the ability to migrate and properly operate at any level of a Fog deployment's hierarchy. Applications should be able to span all levels of a deployment to maximize their value.

There are three identified viewpoints in the Architecture description diagram: Software, System, and Node.

- Software view: Represented in the top three layers shown in the architecture description, and include Application Services, Application Support, and Node Management (IB) and Software Backplane.
- System view: Represented in the middle layers shown in the architecture description, which include Hardware Virtualization down through the Hardware Platform Infrastructure.
- Node view: Represented in the bottom two layers, which includes the Protocol Abstraction Layer and Sensors, Actuators, and Control.

The IEEE has adopted the OpenFog Consortium's OpenFog Reference Architecture, which defines how computing resources and services are distributed between the Cloud, Edge computing facilities

and devices, as an official standard, IEEE 1934 [34]. The OpenFog Consortium relies on the reference architecture as a universal technical framework that enables the data-intensive requirements of the Internet of Things (IoT), 5G and artificial intelligence (AI) applications.

2.4 DATA PROCESSING IN IOT

As has been previously argued, the volume of data already generated by connected devices is vast and needs an approach based on BigData techniques.

There are a huge amount of BigData solutions and infrastructures related to data processing, from batch processing to streaming data and user-facing abstractions. The current focus on BigData place particular emphasis on the power of data and data mining solutions and the technologies solutions are focused on handle these large volumes of data and trying to find patterns and trends from them.

In a IoT environment it is not enough to store information in a data warehouse and report back on its days later. In IoT use cases, data needs to be processed on a streaming basis with the ability to identify and act on interesting information quickly and effectively. It should also support stream processing from the outset and have the capability to deal with low-latency queries against semi-structured data items, at scale. Simply collecting data is only part of the solution. What's key is the ability to combine deep predictive analytics from historical data with real-time events.

Common requirements for an IoT data processing platform are: support for native raw data, support for a variety of workload types, business continuity and security & privacy.

IoT applications produce big datasets that can't be transferred over the Internet to be processed by a centralized public or private datacenter. The datasets flow at a volume and velocity too large and too fast to be processed by a single centralized datacenter and the analytics models and intelligence required to process the datasets are available across distributed locations. A highly scalable IoT processing platform is essential.

2.5 BENCHMARKING FOR IOT

By the time the work presented in this thesis was started, one of the first challenges was the characterization of a platform for hosting IoT workloads in the Cloud (Chapter 3), but at that time there were no tools to perform it. We had to develop our own evaluation system (Section 3.5) in order to achieve this goal. Currently we find Benchmarking tools for IoT like the widely used TPCx-IoT [29]. Here below we will see the description of TPCx-IoT where will be seen that some

of the lines that we follow in our characterization of the performance of our state-of-the-art platform are found in the TPCx-IoT benchmark.

TPCx-IoT is the industry's first benchmark which enables direct comparison of different software and hardware solutions for IoT gateways. The TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. Some of the full members of the TPC corporation are IBM, Intel, Cisco, AMD and Microsoft. TPCx-IoT was specifically designed to provide verifiable performance, price-performance and availability metrics for commercially available systems that typically ingest massive amounts of data from large numbers of devices, while running real-time analytic queries. Provides an objective measure of performance and performance of commercially available software and hardware systems in IoT gateway environments.

The TPCx-IoT workloads consists of data ingestion and concurrent queries simulating workloads on typical IoT Gateway systems. The dataset represents data from sensors from electric power stations and represents data from 200 different types of sensors. The data generated is ingested and persisted into the System Under Test (SUT) and continuously queried to simulate simple analytics use cases. The System Under Test (SUT) represents an IoT gateway system consisting of commercially available servers and storage systems running a commercially available NoSQL data management system.

The workload represents data inject in to the SUT with analytics queries in the background. The analytic queries retrieve the readings of a randomly selected sensor for two 30 second time intervals, TI_1 and TI_2. The first time interval TI_1 is defined between the timestamp the query was started Ts and the timestamp 5 seconds prior to T_S, i.e. $TI_1 = [T_S - 5, T_S]$. The second time interval is a randomly selected 5 seconds time interval TI_2 within the 1800 seconds time interval prior to the start of the first query, $T_S - 5$. If $T_S <= 1810$, prior to the start of the first query, $T_S - 5$.

TPCx-IoT defines the following primary metrics:

- The Performance Metric ($IoTps$) that represents the effective throughput capability of the SUT.
- The Price Performance Metric ($\$$) defined as: $\frac{\$}{IoTps} = \frac{P}{IoTps}$ where P is the total cost of ownership of the SUT.

The benchmark test consists of two runs, Run1 and Run2. Each run consists of a Warmup Run and Measured Run. The total elapsed time for the Performance run, in seconds (T), is used for the Performance Metric calculation. The Performance Run is defined as the Measured Run with the lower Performance Metric. The Reported Performance Metric is the Performance Metric for the Performance Run.

2.6 NFV MANO

Network functions virtualization (NFV) (also known as virtual network function (VNF)) proposes a new approach to the implementation and operation of network functions, and may inspire the development and deployment of new types of network functions. NFV is a network architecture concept that uses the technologies of IT virtualization to virtualize entire classes of network node functions into building blocks that may connect, or chain together, to create communication services. NFV decouples the network functions, such as network address translation (NAT), firewalling, intrusion detection, domain name service (DNS), and caching, to name a few, from proprietary hardware appliances so they can run in software. NFV offers a new way to design, deploy and manage networking services.

It is designed to consolidate and deliver the networking components needed to support a fully virtualized infrastructure - including virtual servers, storage, and even other networks. It utilizes standard IT virtualization technologies that run on high-volume service, switch and storage hardware to virtualize network functions. It is applicable to any data plane processing or control plane function in both wired and wireless network infrastructures. NFV relies upon, but differs from, traditional server-virtualization techniques, such as those used in enterprise IT. A virtualized network function, or VNF, may consist of one or more virtual machines running different software and processes, on top of standard high-volume servers, switches and storage devices, or even cloud computing infrastructure, instead of having custom hardware appliances for each network function.

NFV MANO (network functions virtualization management and orchestration), also called MANO, is an architectural framework for managing and orchestrating virtualized network functions (VNFs) and other software components. The European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG NFV) defined the MANO architecture to facilitate the deployment and connection of services as they are decoupled from dedicated physical devices and moved to virtual machines (VMs). A general view of the MANO architecture can be seen in Figure 2.4. The Network Functions Virtualisation Management and Orchestration (NFV-MANO) architectural framework has the role to manage the NFV infrastructure and orchestrate the allocation of resources needed by the network services and VNFs. Such coordination is necessary now because of the decoupling of the Network Functions software from the NFV infrastructure.

The NFV-MANO architectural framework identifies the following functional blocks:

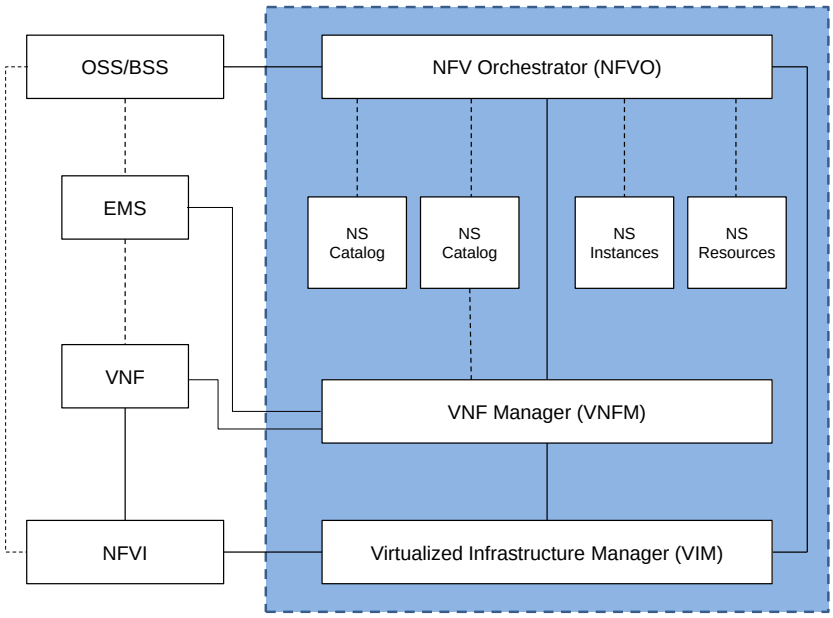


Figure 2.4: NFV MANO Architecture

- NFV Orchestrator (NFVO): Consist of two layers – service orchestration and resource orchestration – which control the integration of new network services and VNFs into a virtual framework. NFV orchestrators also validate and authorize NFV infrastructure (NFVI) resource requests.
- VNF Manager (NFVM): Oversees lifecycle management of VNF instances; coordination and adaptation role for configuration and event reporting between NFVI and Element or Network Management System (EMS/NMS).
- Virtualised Infrastructure Manager (VIM): Controls and manages NFV infrastructure, which encompasses compute, storage and network resources.

MANO works with templates for standard virtual network functions so users can pick from existing network functions virtualization infrastructure resources to deploy their NFV platform. For NFV MANO to be effective, it must be integrated with application program interfaces (APIs) in existing systems in order to work with multivendor technologies across multiple network domains.

2.7 YANG

YANG (Yet Another Next Generation) is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols.

YANG is a language originally designed to model data for the NETCONF protocol. A YANG module defines hierarchies of data that can be used for NETCONF-based operations, this allows a complete description of all data sent between a NETCONF client and server.

YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between those nodes.

The language, being protocol independent, can then be converted into any encoding format, e.g. XML or JSON, that the network configuration protocol supports. YANG is a modular language representing data structures in an XML tree format. The data modeling language comes with a number of built-in data types. Additional application specific data types can be derived from the built-in data types. More complex reusable data structures can be represented as groupings. YANG data models can use XPATH expressions to define constraints on the elements of a YANG data model.

The following Yang module, code1, *l3vpn* shows a partial data model for a layer 3 vpn. The module declares a namespace and a prefix and imports the type library module *ietf-inet-types* before defining the type department. It then defines a container *topology* that includes a list of *roles* and a list of *connections*. A connection has two *endpoints* that reference roles via the *leafref* type and its *path* restriction.

Listing 1: Yang example for a Layer 3 VPN

```
module l3vpn {
    namespace ‘‘http://com/example/l3vpn’’;
    prefix l3vpn;

    import ietf-inet-types { prefix inet; }

    typedef department {
      type string;
      description
        ‘‘The department that needs the VPN’’;
    }

    container topology {
      list role {
        key name;
        leaf name {
          type enumeration {
            enum ce;
            enum pe;
            enum p;
          }
        }
      }

      list connection {
```

```
        key name;
        leaf name {
          type string;
        }
        leaf endpoint-1 {
          type leafref {
            path ''/topology/role/name'';
          }
        }
        leaf endpoint-2 {
          type leafref {
            path ''/topology/role/name'';
          }
        }
        leaf link-vlan {
          type unit32;
        }
      }
    }
    . . . . .
}
```

2.8 MOBILE EDGE COMPUTING - MEC

Mobile Edge Computing is a network architecture paradigm that provides an IT service environment and Cloud computing capabilities at the Edge of the mobile network, within the Radio Access Network (RAN) and in close proximity to mobile subscribers. The aim is to reduce latency, ensure highly efficient network operation and service delivery, and offer an improved user experience. Mobile Edge Computing (MEC) is a new technology which is currently being standardized in an ETSI Industry Specification Group (ISG) of the same name. To align with broader developments in service provider networking, and to increase the strategic value of MEC, the ISG has adopted the ETSI management and orchestration (MANO) model for service orchestration and management.

Based on a virtualized platform, MEC is recognized by the European 5G PPP (5G Infrastructure Public Private Partnership) research body as one of the key emerging technologies for 5G networks with Network Functions Virtualization (NFV). For the evolution of 5G, MEC represents a key technology and architectural paradigm. MEC helps the advance of the transformation of the mobile broadband network into a programmable environment satisfying the demanding requirements of 5G in therms of latency, scalability and automation. A general view of the layer architecture of MEC architecture can be seen in Figure 2.5.

MEC is based on a virtualized platform and has a complementary approach to NFV. NVF is focused on network functions but MEC

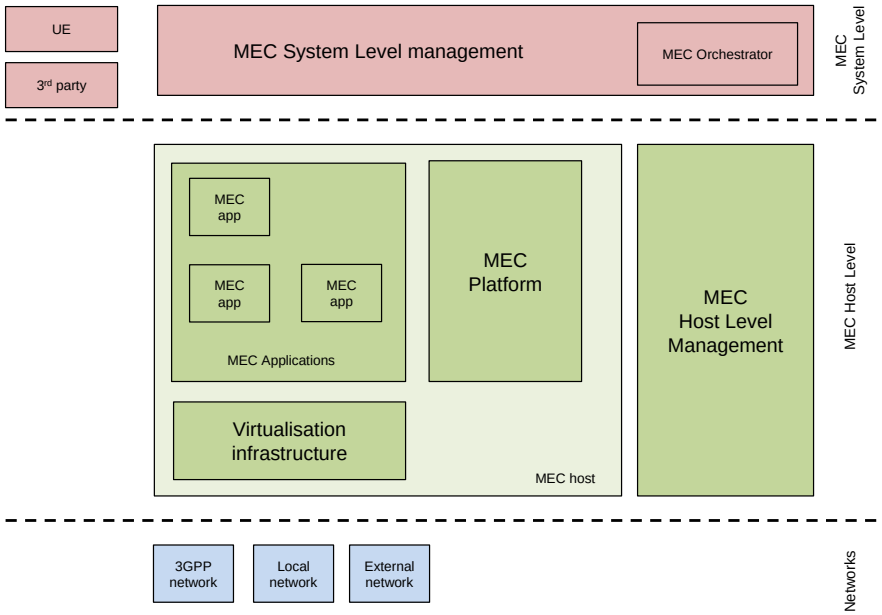


Figure 2.5: MEC Architecture

framework enables applications running at the Edge of the network. There are similarities between the infrastructure that hosts MEC and NFV, so it will be beneficial to reuse the NFV infrastructure by hosting both, VNFs and MEC applications on the same platform. The primary goal of Mobile Edge Computing is to reduce network congestion and improve application performance by achieving related task processing closer to the user.

The major components of the MEC architecture can be described as follows:

- Mobile Edge Platform: Is the component that provides the functionalities that are required to run on the MEC host applications and to enable MEC applications to access the MEC services. It also provides a set of services that expose radio network data and other real time context information to authorized MEC applications.

- Mobile Edge Orchestrator has the visibility over the resources and capabilities of the entire MEC system and maintains an overall view of the deployed MEC servers to determine the locations for instantiating the MEC applications. In many ways it is similar to the ETSI Network Functions Virtualization Orchestrator (Section 2.6).

- Mobile Edge Platform Manager is the responsible for the lifecycle management of the MEC applications and management of the MEC Application Platform and the MEC application pol-

icy management functions. Is also responsible for instantiating, terminating and relocating the MEC applications.

- Virtualized Infrastructure Manager is responsible for managing the resources of the virtualized infrastructure, like releasing and allocating vitualized storage, network resources and compute. And also includes preparing the infrastructure for running a software image.
- Mobile Edge Applications run as virtual machines on top of virtualization infrastructure provided by the MEC host. They must use the MEC application programming interfaces (APIs) and be manageable within the NFV framework.
- MEC host is a logical construct which embraces the MEC platform and the virtualization infrastructure that provides compute, storage and network resources to the MEC applications.

3

HOSTING IOT DATA-CENTRIC WORKLOADS IN THE CLOUD

3.1 INTRODUCTION

In the last years, Internet of Things (IoT) and Big Data platforms are clearly converging in terms of technologies, problems and approaches. IoT workloads generate a vast amount of data that needs to be stored and processed, becoming a Big Data problem. IoT devices and sensors generate streams of data across a diversity of locations and protocols that in the end reach a central platform that is used to store and process these streams. Processing can be done in real time, with transformations and enrichment happening on-the-fly, but it can also happen after data is stored and organized in repositories. In the former case, stream processing technologies are required to operate on the data; in the latter analytics and queries are of common use. This chapter presents a detailed characterization of the of the servIoTicy [24] platform: a state-of-the-art infrastructure for hosting Internet of Things (IoT) workloads in the Cloud. It provides multi-tenant data stream processing capabilities, a REST API, data analytics, advanced queries and multi-protocol support in a combination of advanced data-centric technologies. The deployment of such a complex platform in the cloud requires a detailed understanding of all these components and tiers to allow for auto-scaling and dynamic provisioning capabilities. This chapter aims to provide this initial characterization to be the basis for advanced cloud provisioning strategies and algorithms.

The above mentioned situation implies that there is an increasing demand for advanced IoT data management and processing platforms. Such platforms require support for multiple protocols at the

edge for extended connectivity with the objects, but also need to exhibit uniform internal data organization and advanced data processing capabilities to fulfill the demands of the application and services that consume IoT data.

To provide answer to this growing demand, servIoTicy is a state-of-the-art platform for hosting Internet of Things (IoT) workloads in the Cloud. It provides multi-tenant data stream processing capabilities, a REST API, data analytics, advanced queries and multi-protocol support in a combination of advanced data-centric services. ServIoTicy aims to provide a technological platform for easily creating services based on the Internet of Things (IoT), thus unleashing the full potential of an Internet of Services (IoS) based on the IoT. The main focus of servIoTicy is to provide a rich set of features to store and process data through it REST API, allowing objects, services and humans to access the information produced by the devices connected to the platform. servIoTicy allows for a real time processing of device-generated data, and enables for simple creation of data transformation pipelines using user generated logic. Unlike traditional service composition approaches, usually focused on addressing the problems of functional composition of existing services, one of the goals of the servIoTicy is to focus on data processing scalability. Other components that can be connected to servIoTicy provide added capabilities to automatically create compositions of high-level services using existing tools [63]. Figure 3.1 provides an overall view of the features offered by servIoTicy.

Advanced streaming and analytics platforms such as servIoTicy are complex pieces of software that integrate a large set of components under the hood. They hide their complexity behind simple REST APIs and multi-protocol channels, but the reality is that their deployment and configuration is complex. Automatically provisioning resources in the cloud for hosting these platforms as a service requires a detailed understanding of all these components and tiers to allow for auto-scaling and dynamic provisioning capabilities. This chapter aims to provide this initial characterization on servIoTicy to be the basis for advanced cloud provisioning strategies and algorithms. The workload characterization described in this chapter also offers an interesting insight for anybody interested in understanding the resource demands of modern indexing platforms such as ElasticSearch [10] or distributed data stores such as CouchBase [4].

The platform described and characterized in this chapter is part of the developments of the COMPOSE [55] project, which aims to develop a more ambitious IoT platform, not only focused on the data management and processing part, but including other aspects such as security, discovery of objects, development tools and composition engines. The sources of the servIoTicy are freely available as an open

source project[1] in GitHub. The platform is also available for single node testing as a vagrant box, downloadable from a github repository[2].

The main contributions of this chapter are:

- A detailed workload and resource characterization of the servIoTicy major components (REST API tier, Distributed Data Store and Indexing Engine) from a point of view of scalability with the available resources (Experiment 1) and with the load (Experiment 2).
- An evaluation of the efficiency of CouchBase as a distributed data store in terms of response time delivered wit the load (Experiment 3)
- An insight on the performance impact of a proper configuration of the memory settings in ElasticSearch (Experiment 4).

The remaining sections of the chapter are structured as follows: Section 3.2 introduces a set of abstractions defined in servIoTicy for managing data associated to objects; Section 3.3 introduces the general architecture and components of the platform; Section 3.4 describes the main features of the REST API of the platform and its associated data models; Section 3.5 presents the evaluation methodology and the four experiments included in the chapter; Finally, Section 3.6 goes through the related work and Section 3.7 provides a summary of the chapter.

3.2 ABSTRACTIONS USED IN SERVIOTICY

Several abstractions are used in servIoTicy to embrace the different entities involved in the existence of IoT ecosystems. Figure 3.1 provides a visual representation of the features and abstractions provided by servIoTicy, that are described in more detail in this section.

- Web Object: The IoT is composed of objects, either connected to the Internet or not. The group of objects not directly connected to the Internet (e.g. a bottle of wine with a RFID or NFC tag) will need a proxy to represent them in the servIoTicy. There is also a group of objects which may have network capabilities, but limited programmability and support for advanced network protocols. These devices, such as simple sensors, still will need the use of proxies to be able to communicate with servIoTicy. Finally, there is a group of advanced devices (so-called Smart Objects, such as a Smart Phone, tablet, or an Arduino device)

1 https://github.com/serviotic y
2 https://github.com/serviotic y/serviotic y-vagrant

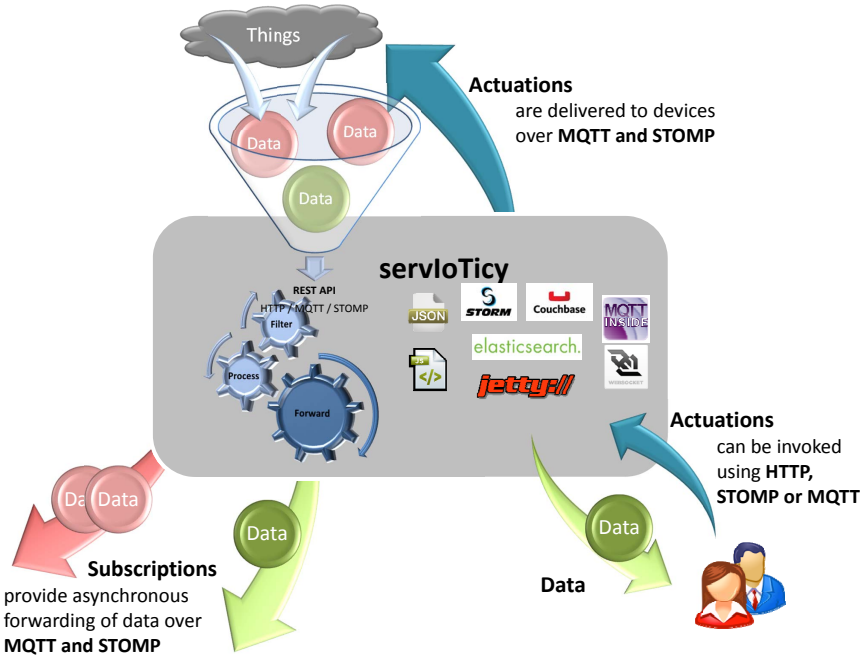


Figure 3.1: servIoTicy features

that already hold the capabilities to talk to the COMPOSE platform directly. Each one of the above mentioned objects (enabled with a communications proxy when needed) is known as a Web Object in servIoTicy. Web Objects are physical objects sitting on the edge of the servIoTicy and capable of keeping for example HTTP-based bi-directional communications, such that the object will be able to both send data to the platform and receive activation requests and notifications. Not all such objects will support the same set of operations, but a minimum subset will have to be guaranteed to make them usable to servIoTicy.

- Service Object: Service Objects are standard internal servIoTicy representations of Web Objects. servIoTicy specifies an API (described in detail in Section 3.4) by which it expects to communicate with the Web Objects, in order to obtain data from them, or set data within them. That API can be embedded directly in the Objects or can be provided by a mediating proxy that will connect the Objects to their corresponding servIoTicy Service Objects. This entity serves mainly for data management purposes and has a well-defined and closed API. That API is needed in order to streamline and standardize internal access to Service Objects, which can in turn represent a variety of very different Web Objects providing very different capabilities. ServIoTicy, in an effort to embrace as many IoT transports as possible, allows Web Objects to interact with their representatives in the Platform (the Service Objects) using a set of well-known protocols: HTTP, STOMP [26] over TCP, STOMP over WebSockets [30], and MQTT [20] over TCP.

- Data Processing Pipe: A Data Processing Pipe is a data service and aggregation mechanism, which relies on the data processing and management back-end component to provide complex computations resulting from subscriptions to different Service Objects as data sources. This construct can support pseudo-real time data stream transformations, combined with queries concerning historical data. Data analytics code defined by the user may be provided as well. The end result of a Data Processing Pipe is inserted into the servIoTicy registry along with its description and may be used by higher level constructs as yet another kind of Service Object building block. Just like a Service Object, this entity serves mainly for data management purposes and has a well-defined and closed API.

- Subscription: Data subscriptions are a mechanism in servIoTicy that allow Service Objects, Data Processing Pipes and external data consumers to get data updates automatically and asynchronously forwarded for further processing.

3.3 ARCHITECTURE OF SERVIOTICY

A general view of the servIoTicy architecture can be seen in Figure 3.2. The Front-End of platform is a Web Tier that implements the REST API that sits at the core of servIoTicy. The API contains part the logic of the Service Objects and Data Processing pipes, related to authentication, data storage and data retrieval actions. The Stream Processing Topology is responsible for the execution of the code associated to Data Processing pipes as well as the forwarding of data across Service Objects and to external entities (e.g. external subscribers that want data forwarded on real-time using a push model on top of MQTT or STOMP). Finally, the data Back-End includes the Data Store that provides scalable, distributed and fault-tolerant properties to servIoTicy, and the Indexing Engine that provides search capabilities across sensors data using different criteria, like timestamps, string patterns or geo-location. In this Section we describe in more detail the main properties of each component of the servIoTicy architecture.

We describe in more detail the main properties of each component of the servIoTicy architecture related to this thesis.

3.3.1 *Web Tier*

The Web Tier for the REST API is composed of a Servlets Container and a REST Engine. As a HTTP Web Server and Java Servlet container we use Jetty [18], which is a pure Java-based HTTP server and Java Servlet container. Jetty is often used for machine-to-machine communications, usually within larger software frameworks. As a

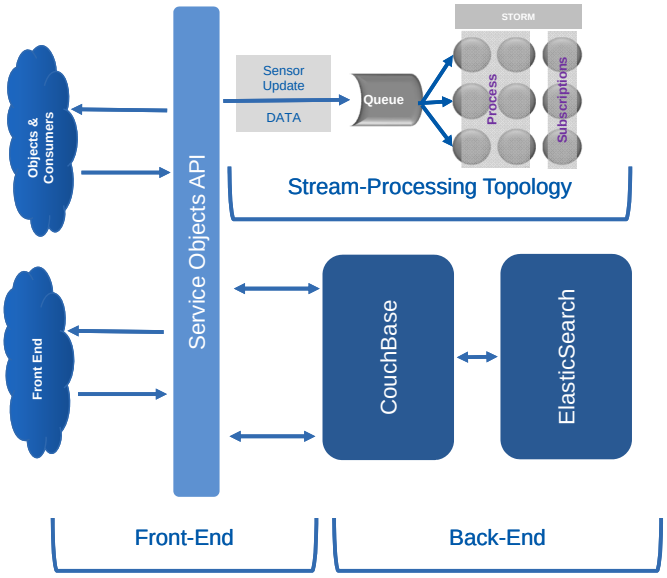


Figure 3.2: ServIoTicy architecture diagram.

REST Engine (JSON processor) we use Jackson [17], which is a high-performance suite of data-processing tools for Java, including the flagship JSON parsing and generation library, as well as additional modules. The Jackson Project also has handlers to add data format support for JAX-RS implementations like Jersey.

3.3.2 *Data Store*

A distributed data store is used to keep track of all the object produced data. For that purpose, CouchBase [4] has been chosen because it provides the benefits of NoSQL data stores (highly distributed, high-availability properties, scalable), and it is document oriented (which fits well for many different data sources and formats). Couchbase has native support for JSON documents. Each JSON document can have a different structure, and multiple documents with different structures can be stored in the same CouchBase bucket. Document structure can be changed at any time, without changing other documents in the database. A Bucket is defined as the owner of a subset of the key space of a Couchbase cluster. These Buckets are used to distribute information effectively across a cluster. A Bucket is equivalent to a database. A common practice is to store For the work presented in this paper we have defined two buckets. One to store the Service Objects definitions and another to store the data associated to a Service Objects data streams.

3.3.3 *Data Indexing*

Queries on the data associated to Service Objects are available using a query DSL. The mechanism to send queries to the platform has been integrated in the API used to access Servioticy. The search infrastructure to resolve queries is provided by an underlying component that performs high-performance indexing and search operations. In particular Elasticsearch [10] is leveraged as it is one of the most powerful and extended search engines that can be integrated with scalable data back-ends (in particular Couchbase). The integration between Couchbase and Elasticsearch enables full-text search, indexing and querying and real-time analytics for variety of use cases such as a content store or aggregation of data from different data sources.

3.3.4 *Stream Processing Topology*

The Stream Processing Topology is implemented on top of Apache STORM [27], which is a state-of-the-art stream processing runtime. Out-of-the-box, STORM provides the availability to build topologies composed of spouts (sources of data) and bolts (processing units). Topologies are static after their deployment, and data keeps flowing through their bolts until the topology is stopped. In case that a different topology is needed, the user needs to stop the running topology and deploy the new one. ServIoTicy implements a novel dynamic user-code injection mechanism that allows STORM bolts to run different versions of code depending on the piece of data that is being processed. The result is that one single STORM topology can in turn be running multiple virtual topologies on top of it, being defined by user preferences. This technique transforms the STORM topology in a multi-tenant data stream processing platform. The mechanism is out of the scope of this paper, and no further details will be provided about it in the context of this work. The Stream Processing Topology also requires the support of a queuing system that will act as the spout for the STORM topology. In servIoTicy, this is implemented using Kestrel [19].

3.4 SERVIOTICY API

Service Objects are exposed through a RESTful API that uses HTTP as a transport and that acts as the SO front-end. This basically implies that SOs can be identified unambiguously using uniques URIs. The API provides resource actuations through the four main HTTP operations: GET (retrieve), POST (create), PUT (update) and DELETE. Table 3.1 contains a summary of the operations implemented in the

REST API. A more detailed description can be found in [65] as well as online [3].

SOs are created by POSTing a JSON document to the API. The document is a basic description of the main properties of the Service Object about to be created. The following example illustrates the case of a SmartPhone object enabled with three different sensors (GPS location, Microphone and Temperature Sensor), each one becoming a stream of data in the SO abstraction. The device is also presenting the capability to be activated through the platform: when the *vibrate action* is invoked on it, the device will vibrate to notify something to the user carrying it.

The corresponding response message contains the field `"id"` that is the unique identifier of the SO within the COMPOSE platform (the `<soId>` The response also contains the list of the `<streamsId>` in the field `"streams"`.

As discussed before, a SO update is actually a JSON document containing, among other information, a tuple of values that correspond to the channels of a device sensor, what is represented as a *stream* in the SO representation.

Listing 2: Format of a Sensor Update

```
{ "channels": [
    { "name": "temp",
      "current-value": 22.58,
      "type": "numeric",
      "unit": "m/s2"
   } ],
    "name": "temperature",
    "lastUpdate": 194896800,
    "customFields": {
        "covered-period": "24h",
        "averageLastHour": 32,
        "risk": "low",
        "averageLastDay": 42
    }
}
```

An example of pushing data from the SmartPhone to its corresponding SO counter-part is achieved by submitting a JSON document to the corresponding url. The `<soId>` would be obtained from the previous creation of the SO. `streamId` should be picked from the list of streams existing in the SO description. In this example, temperature data is being pushed to the platform. As it can be observed in the following example, the information for the *temp* channel, which is associated to the temperature stream, includes the actual temperature value as well as other information such as units, update time

3 http://docs.servioticy.com

operation	*Target URI*	*Role*
Create	POST /	Create a new SO posting a JSON document.
Retrieve	GET /	Retrieve the list of all the SOs created.
Retrieve	GET /<soId>	Retrieve attributes from the <SO_Id> Service Object.
Update	PUT /<soId>	Modify the <SO_Id> ServiceObject.
Delete	DEL /<soId>	Delete the <SO_Id> Service Object.
Retrieve	GET /<soId>/streams	Retrieve the list of all the SO streams.
Create	POST /<soId>/streams/<streamId>/subscriptions	Subscribe the Service Object <soId> to a service .
Update	PUT /<soId>/streams/<streamId>	Store <soId> data puting a JSON document.
Retrieve	GET /<soId>/streams/<streamId>	Retrieve the list of all the data of <soId> Service Object.
lastUpdate	GET /<soId>/streams/<streamId>/lastUpdate	Retrieve the last piece of data generated by a <soId> Service Object.

Table 3.1: API operations

and a series of custom fields that the WO can decide to add when generating the SU for future reference.

To retrieve and delete a SO, the corresponding operation can be invoked for each <soId> URL. For the former a response JSON document describing the success of the operation is generated, as well as a 200 HTTP status code. For the latter, a new SO description must be associated to the PUT HTTP operation on the <soId> URL, resulting in the SO description being updated and a response generated analogously to the SO creation case described above.

Finally, retrieving data associated to a SO stream results in a JSON document containing an array of all the tuples stored for this stream.

The creation of subscriptions is at the core of the COMPOSE platform and allows for the definition of data processing paths that are followed by SUs being generated by external WOs and afterwards ingested by the platform. Subscriptions can be internal (when one SO wants to get SUs generated by other SOs to be forwarded to it), or external, when entities outside of the COMPOSE platform wants to be notified about any SUs produced by one particular SO.

3.5 EVALUATION

After the description of the platform we will move to the evaluation of its performance and the viability of the proposed solutions. The goal of our evaluation was to explore the following questions:

- Experiment 1: How does throughput and response time scale for the servIoTicy API with the amount of available resources?
- Experiment 2: How does throughput and response time scale for the servIoTicy API when the load increases?
- Experiment 3: How does the response time delivered by CouchBase scale with the API pressure?
- Experiment 4: How does ElasticSearch scale with the load and how does its configured heap size affects its performance?

To provide answers to these questions a set of four different experiments have been developed that are presented in the following subsections.

3.5.1 *Evaluation Methodology and Infrastructure*

For the client emulation we used Httperf [58] specifying a session workload generator, designed to simulate a real users progress through a site. This type of testing is useful for estimating the actual performance that a web server or an API will achieve in practice. A file was created containing the sequence of requests to be performed, the number and sequence of URI's and request method. We created a file containing the list of all existing Service Objects, randomly sorted and not repeated more than 8 times.

For such purpose, 6 parallel Httperf processes were used, each one emulating 300 clients (at which 100 new sessions were created per second) for a total of 1800 clients, each one issuing requests at a variable rate. It was verified that each Httperf process had no internal bottlenecks. The target request rate was changed from 1 request per second to 40 requests per second, resulting in an overall target load of 1,800 to 72,000 requests per second.

Requests are distributed across different Service Objects a specified before, and the in all cases the API call to perform was 'lastUpdate' (see Table 3.1 for more details). This call is interesting because it involves a query to the search engine to retrieve the latest timestamp for all the stored updates for a given sensor, and then the actual retrieve of the sensor update that is returned to the client. This operation actually is more complex than updates from the point of view of the API: updates are a difficult task for the Stream Processing Topology, that is out of the scope of this paper, but involve less work for the API.

ServIoTicy was populated for the experiments with 52,388 Service Objects and 261,940 Sensor Updates, an average of 5 Sensor Updates per Service Object.

The tests have been run on two sets of nodes: one set for running the client emulators and one set for running the servers of the system under test. The 'server' set was composed of 16 two-way 4-core Xeon L5630 @2.13GHz Linux boxes, for a total of 8 cores per node and 16 hardware threads because hyperthreading was enabled. Each 'server' machine was enabled with 24GB of RAM. The 'client' set was composed of 2 two-way 6-core Xeon E5-2620 0 @2.00GHz Linux boxes, for a total of 12 cores per node and 24 hardware threads because hyperthreading was enabled. Each 'server' machine was enabled with 64GB of RAM. All nodes were connected using GbE links to a non blocking 48port Cisco 3750-X switch.

For the software stack, all nodes were running Ubuntu 12.04LTS, and we used Httperf v0.9.1, nginx 1.2.8, Jetty v9.2.3, Jackson v2.3.1,

Jersey v1.18, Couchbase 2.2.0 Enterprise Edition, Elasticsearch 1.3.4, and Couchbase Transport Plugin for Elasticsearch v2.0.0.

3.5.2 *Experiment 1: Scalability of the API with available resources*

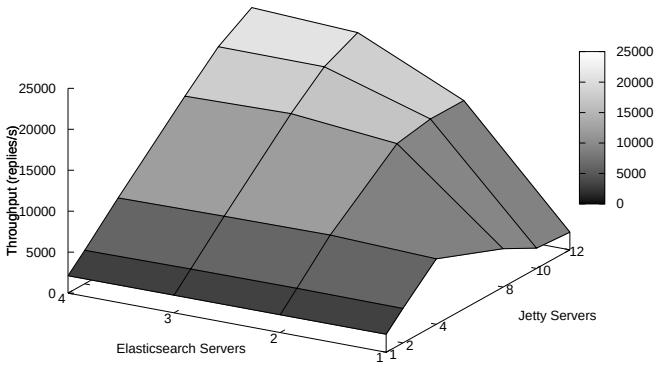
The goal of this experiment is to analyze how the throughput and the response time scale for the servIoTicy API when the number of instances for different servIoTicy components varies. To this end we vary the number of Jetty nodes that serve the API and the number of Elasticsearch servers that perform the request search. We run the same workload changing the request rate target from 1 request per second to 40 requests per second and compare the results. Notice that as this is a closed loop system, request rate target is not always achieved as the response time delivered by the serves will influence the rate at which the client emulators will issue requests.

For this experiment we explore the scalability of the web tier of the IoT REST API for a combination of 1 to 12 Web Servers (Jetty) and a combination of 1 to 3 ElasticSearch servers. A static configuration of 2 CouchBase servers, continuosly synchronized with the ElasticSearch tier are used also.

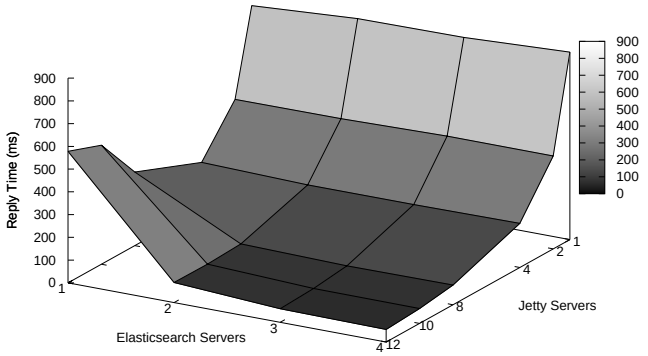
Figure 3.3a shows the results for highest target load explored, a request rate of 40 requests per second each client, resulting in an overall request rate target of 72,000 requests per second. As it can be observed, the Web Tier offers almost linear scalability as the resources committed to this tier increase. This can be observed for the case in which 3 ElasticSearch servers are allocated. This can be considered an expected result because the Web Tier is stateless and does not benefit from any kind of session affinity, what could result in a performance impact for some conventional Web Applications, but this it not the usual case for REST APIs.

As the number of ElasticSearch servers is reduced, from 3 to one, it can be observed how the indexing tier gradually becomes the bottleneck, dropping the overall performance of the servIoTicy API from around 25,000 requests processed per second for 12 Jetty servers and 3 ElasticSearch nodes, to roughly 2,500 requests processed per second inthe case that the same 12 Jetty servers are allocated for the Web Tier and only one instance of ElasticSearch is used.

Looking at the results for response time scalability, shown in Figure 3.3b, it can be clearly observed the same behavior, what is expected for a closed loop system as our testing platform is. In this case, it can be seen how the platform can easily achieve a baseline response time of around 100ms per request for any configuration of the Web tier that is provisioned with 8 or more Jetty servers. But when the number of ElasticSearch servers is varied, from 3 instances to 1 instance, we observe again how the delay introduced per request in the indexing layer starts increasing, increasing substantially the response



(a) Throughput



(b) Response Time

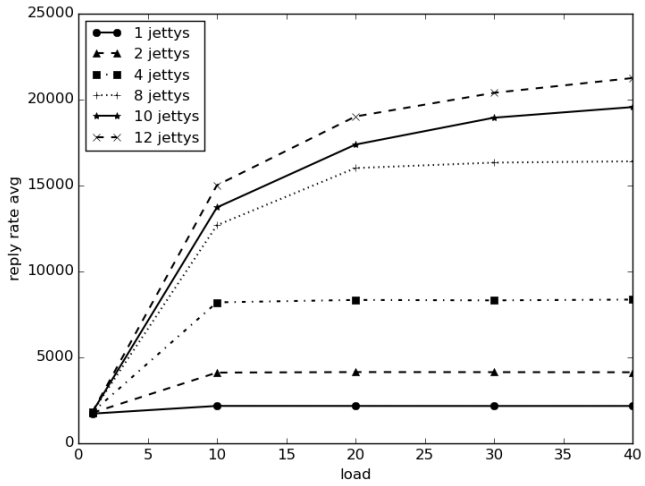
Figure 3.3: API scalability with variations of the number of instances for the Web Tier and ElasticSearch

time and therefore reducing the maximum throughput delivered by servIoTicy as it had been seen in Figure 3.3a.

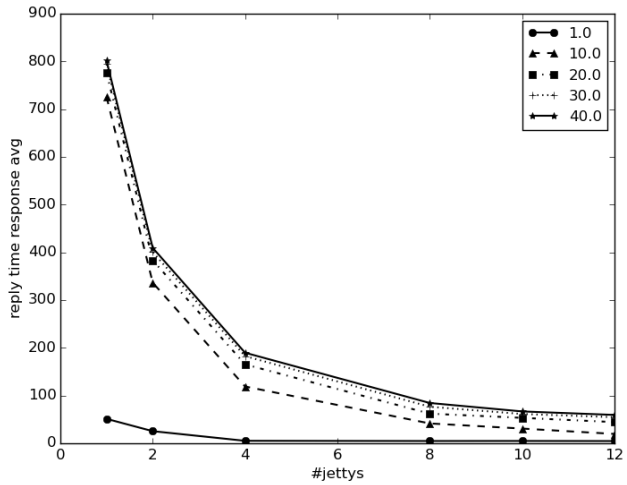
The reason why the experiment does not explore a different number of CouchBase servers is that we have observed that the data tier is not a bottleneck in most situations. CouchBase seems to deliver very low response time for a very high number of requests per second, and therefore the usual case is that either the ElasticSearch tier or the Web tier are responsible for performance bottlenecks. For a better understanding of this statement we refer the reader to Experiment 4 below in this Section.

3.5.3 *Experiment 2: Scalability of the API with load*

Once the scalability of the platform with the allocated instances for the Web and Indexing tier has been explored, we look now in more detail how the throughput and response time scale for servIoTicy with load variations. To do this we set the number of Elasticsearch instances to a fixed number, and we modify the target request rate in the range that goes from from 1 request per second to 40 request per second and client.



(a) Throughput



(b) Response Time

Figure 3.4: API scalability with variations of the number of instances for the Web Tier and the Load Level

As in Experiment 1, we then vary the number of Jetty servers serving the API. In particular, and based on the results obtained for Experiment 1, we fix the number of Elasticsearch instances to three, as it provides a good compromise between scalability and resource usage,

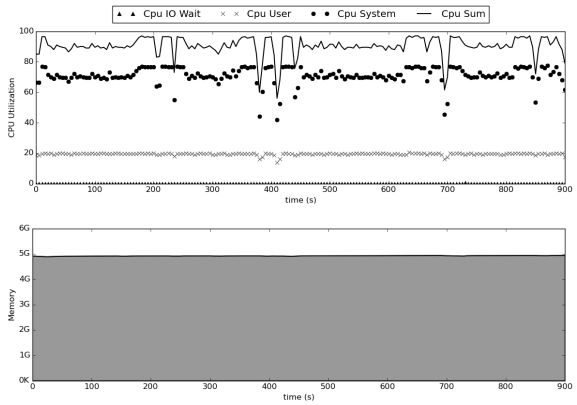
and ensures that ElasticSearch will not be the bottleneck for most of the load levels to be tested.

Figure 3.4a and Figure 3.4b show respectively the throughput and response time associated with each number of Jetty servers for a range of load levels. As it can be observed, the more instances are provisioned for the REST API tier, the higher the maximum achieved throughput is, as more instances are operating in parallel and therefore more requests per second can be processed. A similar behavior can be observed for the response time: there is a baseline response time of about 100ms that is only achievable for those load levels and configurations that are not saturated. Any configuration that provides a too low number of instances for the Web Tier as to keep the equilibrium of the system, quickly results in a system with growing waiting queues for the requests and ends up with increased response times.

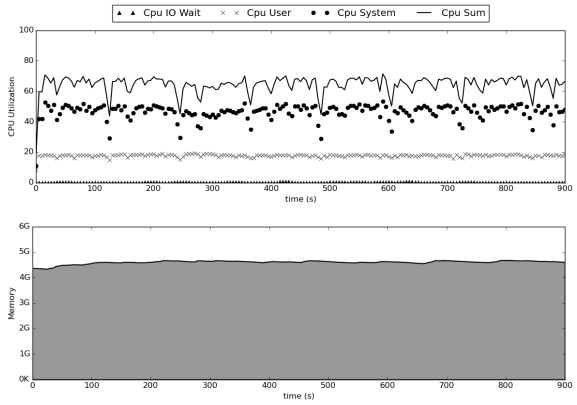
Looking at Figure 3.4a, and taking a particular load level, some interesting scalability properties can be observed. Take for instance load level 40 as an example. It can be seen how for this load level, the system is initially constrained by the capacity of the Web Tier. This fact can be observed because as more instances are added to the Web Tier, the system perfectly scales-out, going for instance from around 2,000 replies per second for a load level 40 and 1 Jetty instance to around 4,000 replies per second when the capacity of the Web Tier is doubled to two instances. This behavior is generally observed until other limits are reached. An example of this situation can be seen for the same load level and 10 or 12 instances allocated for the Web Tier. In these scenarios, the scalability exhibited by the system is not linear, and therefore, other limiting factors must be reach.

To understand the situation, we monitor the resource consumption of the platform components for different load levels. Figure 3.5 shows the average CPU and memory consumption for the Web Tier instances (the cases in which 2 instances and 12 instances are provisioned) and for the combination of ElasticSearch and CouchBase components, when the load level is 40. As it can be observed, for the case in which 2 Jetty instances are provisioned(Figure 3.5a) shows how the servers hosting those instances are clearly overloaded, with a total CPU consumption above 90% all the time. When we compare these numbers with the resource consumption observed for ElasticSearch and CouchBase (Figure 3.5c), it can be clearly seen how those components are not suffering from the same overload. For the case in which 12 Jetty instances are used, the situation is the opposite, showing a lower CPU utilitzation for the Web Tier (Figure 3.5b) than the data tier (Figure 3.5d).

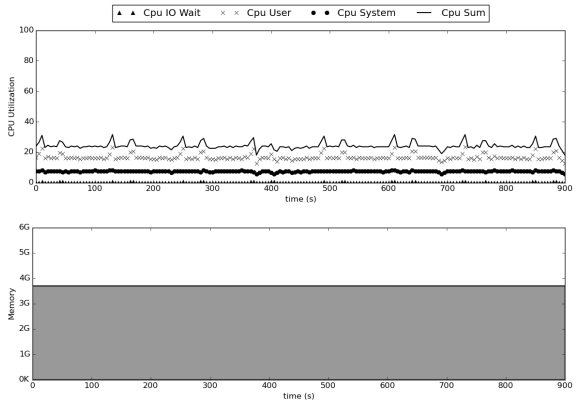
Notice that in all cases, memory is not a limiting factor for the tests included in this experiment. Nevertheless, this is not always the situation, and we refer the reader to Experiment 4 for an example of effects of memory consumption on the performance of the platform.



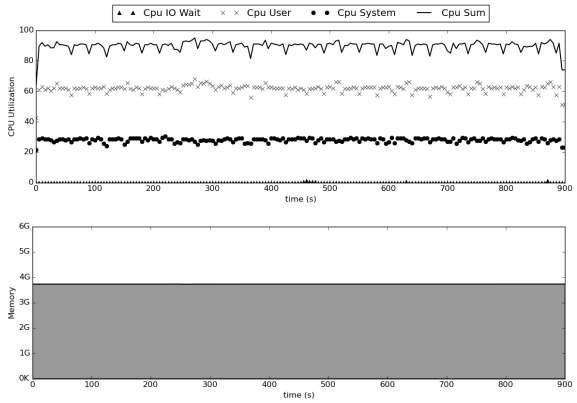
(a) 2 Jetty Instances - Jetty



(b) 12 Jetty Instances - Jetty



(c) 2 Jetty Instances - ES + CB

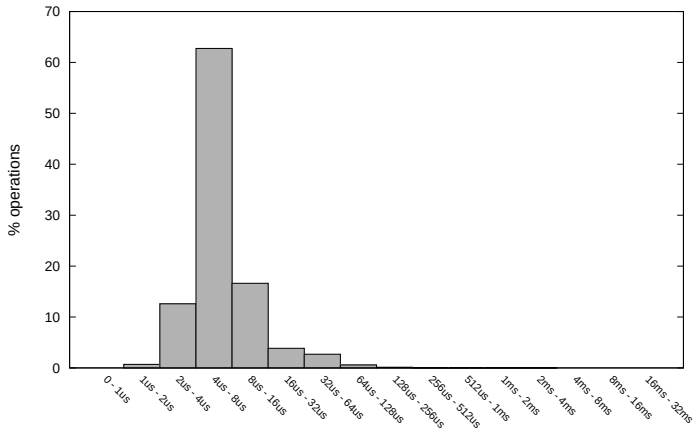


(d) 12 Jetty Instances - ES + CB

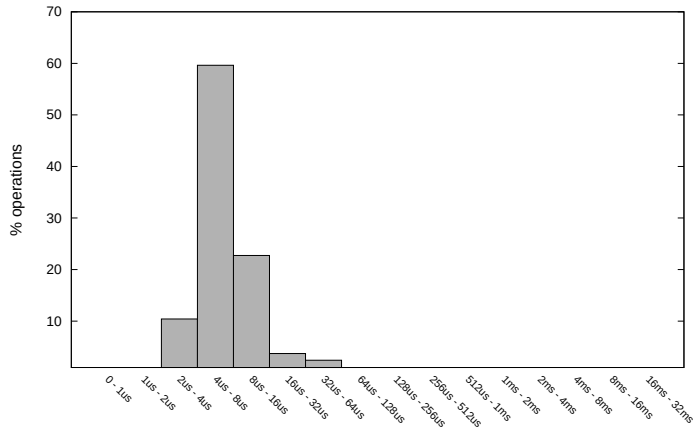
Figure 3.5: Resource consumption (CPU, Memory) for the REST API Web tier (Jetty) - Load Level 40

3.5.4 *Experiment 3: Scalability of the Data Store*

The goal of this third experiment is to evaluate the horizontal scalability of the Couchbase tier and its impact in servIoTicy performance. The motivation for this experiment is the fact that as we went through all the tests included in this paper, we realized that CouchBase was delivering very low response times independently of the number of instances that we were provisioning. Even 12 Jetty instances for the Web Tier were apparently unable to saturate a single instance of Couchbase. To put some light on the reponse times delivered by Couchbase on our experiments and to understand by how much this tier was responsible for the baseline response time observed for servIoTicy, we built the response time histograms that can be observed in Figure 3.6.



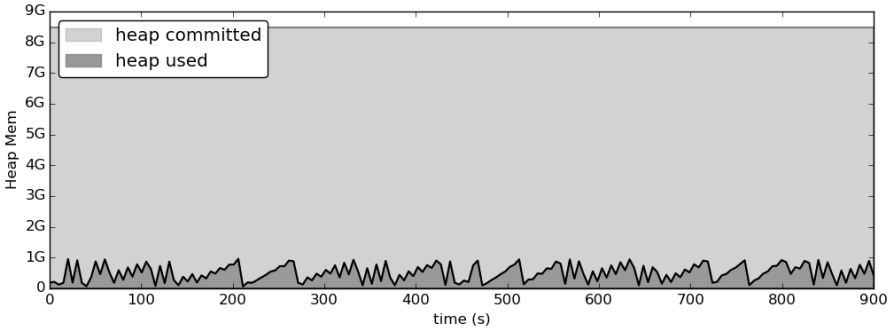
(a) 2 CouchBase Instances - Load 40



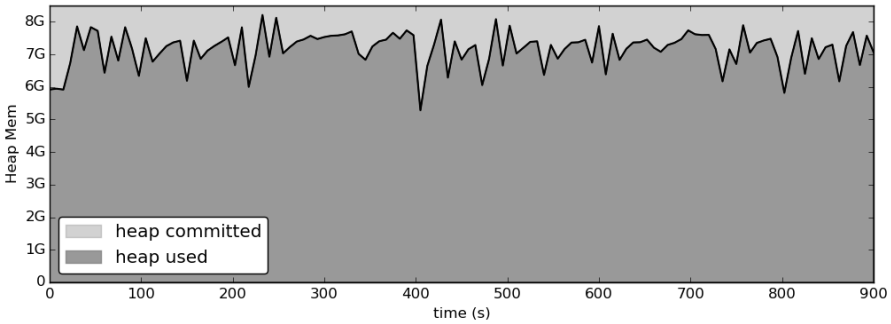
(b) 3 CouchBase Instances - Load 40

Figure 3.6: Distribution of response times (bins) - Couchbase tier

In this experiment take the more demanding load level (40), we provision 12 Jetty servers for the Web Tier and we compare the behavior of the Couchbase tier when two and three instances of Couchbase



(a) 1 Jetty Instance



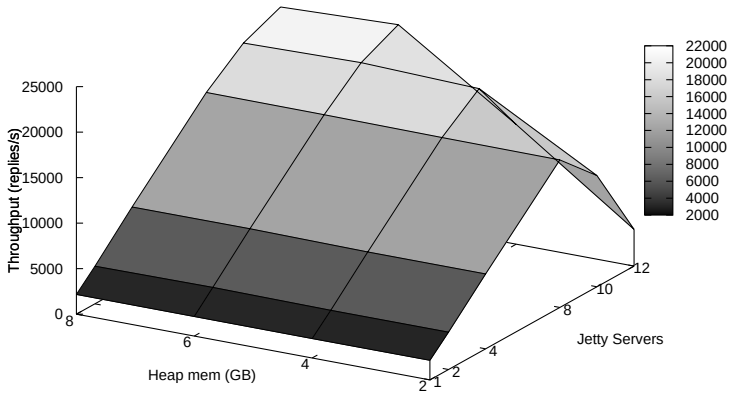
(b) 12 Jetty Instances

Figure 3.7: Variation of memory Heap usage for Load Level 40, one ElasticSearch instance and a variation of instances provisioned for the Web Tier

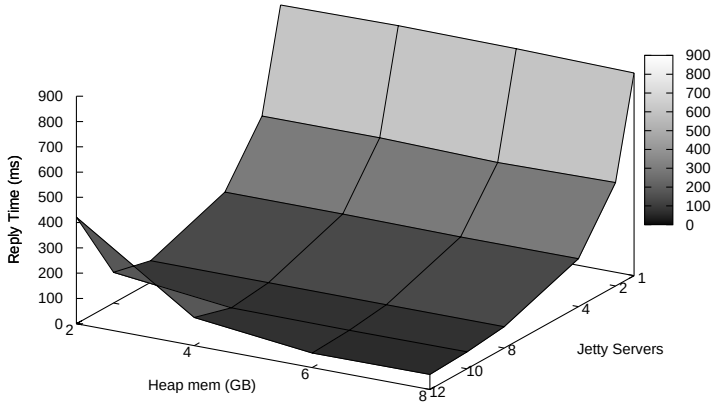
are provisioned. Results are organized in response time bins on the x-axis, and in the y-axis we show the percentage of the total responses generated in the experiment that fell in that bin. There are two facts to remark in this experiment: First, the confirmation that Couchbase is delivering very low response times, with all the requests being served in less than one millisecond; Second, that when we compare the response time delivered by two instances against the observed times for three instances, it can be seen that no difference is noticeable, indicating that the instances deliver perfect horizontal scalability and that very rarely they will become the bottleneck for servIoTicy deployments.

3.5.5 *Experiment 4: Performance limitations of the Indexing components*

In this experiment we evaluate how ElasticSearch performance is affected by the Heap Size in that is configured in Java when running the indexing instances. For this purpose we look in detail the execution of the servIoTicy workload this time using a single Elasticsearch instance and a load 40.



(a) Throughput



(b) Response Time

Figure 3.8: API scalability with variations of the memory Heap configuration for a set of three ElasticSearch instances, a variation of the instances provisioned for the Web Tier and a Load Level 40

The first thing to explore is how the memory Heap consumption is affected by the load level that reaches the ElasticSearch tier in servIoTicy. Figure 3.7a and 3.7b show the heap memory overhead produced for a load of 40 when the Web Tier is provisioned with 1 and with 12 Jetty servers respectively. As it can be observed, when 1 single Jetty instance is provisioned, as it acts as a bottleneck for the platform, the load that reaches the ElasticSearch tier is low and therefore the memory Heap utilization is low. In contrast, when 12 Jetty instances are provisioned and a much higher demand reaches the indexing tier, the Heap utilization becomes extremely high and, as it will be show below, this situation results in a significant performance degradation.

To quantify the peformance impact of memory Heap configuration on ElasticSearch, we set the number of Elasticsearch instances to three and we analyze for a load level 40, the impact of different Heap memory configurations (ranging from 2GB to 8GB per instance) as we vary the number of Jetty servers. Figure 3.8a and Figure 3.8b show the throughput and response time delivered by servIoTicy under these configurations. As it can be observed, the capacity of the system to deliver sustained performance is seriously affected by the memory allocated to the ElasticSearch instances. As an example, the same number of Jetty and ElasticSearch instances, using 8GB of Hep per indexing instance allows servIoTicy to deliver up to 25,000 replies per second, while changing the Heap size to 2GB produces a drop on performance that results in less than 5,000 replies per second.

3.6 RELATED WORK

Few studies present both a characterization of workload and resource consumption for web applications. In [62] Patwardhan et al. perform a CPU Usage breakdown of popular Web benchmarks with emphasis on networking overhead, identifying that network overhead for dynamic applications is negligible, while not for static content. In [87] Ye and Cheng present a similar characterization of resource utilizations as the one presented here, but for *Online Multiplayer Games*. The work presented in this paper is, to our knowledge, the first performance characterization of a Data Centric IoT platform for the Cloud.

Deployment of IoT platforms on the Cloud is also covered in the literature. In [41], authors propose strategies for deciding the best approach at the time of making cloud-based deployments of IoT applications using nowadays regular cloud technologies. Another recent work [37] studies the implementation of IoT platforms on top of cloud-based pub/sub communication infrastructures. Finally, authors go one step beyond in [59] by leveraging completely Software Defined Environments for managing the Cloud infrastructures in which IoT applications are deployed.

Data Centric view of the IoT is not something new for servIoTicy as it was widely covered in the survey presented in [68]. What servIoTicy uniquely provides is an open source solution that challenges the features of commercial solutions such as Xively [31] and Evrythng [13], while extending their capabilities with the ability to inject user-defined code into its stream processing runtime.

There are other open source platforms for IoT in the market, but they are focused on other aspects of the Internet of Things. The DeviceHive [8] project offers a machine-to-machine (M2M) communication framework for connecting devices to the Internet of Things. It includes easy-to-use Web-based management software for creating networks, applying security rules and monitoring devices. Device-

hub.net [9] is a cloud-based service that stores IoT-related data, provides visualizations of that data and allows users to control IoT devices from a Web page. The IoT Toolkit [16] project provides a variety of tools for integrating multiple IoT-related sensor networks and protocols. The primary project is a Smart Object API, but it also aims to develop an HTTP-to-CoAP Semantic mapping. Mango [9] is a popular open source Machine-to-Machine (M2M) software, which is web-based and supports multiple platforms. Key features include support for multiple protocols and databases, and user-defined events among others. Nimbits [21] can store and process a specific type of data previously time- or geo-stamped. A public platform as a service is available, but it can also be downloaded and deployed on Google App Engine, any J2EE server on Amazon EC2 or on a Raspberry Pi. OpenRemote [23] offers four different integration tools for home-based hobbyists, integrators, distributors, and manufacturers. It supports dozens of different existing protocols, allowing users to create nearly any kind of smart device they can imagine and control it using any device that supports Java. The SiteWhere [25] project provides a complete platform for managing IoT devices, gathering data and integrating that data with external systems. SiteWhere releases can be downloaded or used on Amazon's cloud. It also integrates with multiple big data tools, including MongoDB and ApacheHBase. Finally, ThingSpeak [28] can process HTTP requests and store and process data. Key features of the open data platform include an open API, real-time data collection, geolocation data, data processing and visualizations, device status messages and plugins.

3.7 SUMMARY

This chapter presents a detailed characterization of the resource demand observed for the different components of the servIoTicy platform. ServIoTicy is a state-of-the-art platform for IoT services, that integrates multi-protocol channels to communicate with the platform on the edge and data management and processing capabilities at its core. The characterization has revealed interesting details about the three main components involved in the process of storing and retrieving data: the REST API (implemented using Jackson on Jetty), the data store (CouchBase) and the search and indexing engine (ElasticSearch). We have observed how the REST API is generally the bottleneck, clearly CPU-bound, being ElasticSearch the second component more demanding in terms of CPU resources. CouchBase has delivered impressive performance and very low response times across different configurations, allowing one single instance of this data store to fulfill the demand in terms of requests per second of up to 12 API instances. ElasticSearch has shown to be very sensitive to memory configurations: degradation on its performance could be observed

both for lack of free CPU resources as because of a limited amount of available memory. This work is a first stage toward the construction of resource allocation policies for highly distributed data-centric platforms such as servIoTicy.

The work described in this chapter is a summary of the following main publications:

[65] Juan Luis Pérez, Álvaro Villalba, David Carrera, Iker Larizgoitia, and Vlad Trifa. The COMPOSE API for the internet of things. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 971–976. ACM, 2014. ISBN 978-1-4503-2745-9. doi: 10.1145/2567948.2579226. URL http://doi.acm.org/10.1145/2567948.2579226

[64] Juan Luis Pérez and David Carrera. Performance characterization of the servioticy API: an iot-as-a-service data management platform. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 62–71. IEEE Computer Society, 2015. ISBN 978-1-4799-8128-1. doi: 10.1109/BigDataService.2015.58. URL https://doi.org/10.1109/BigDataService.2015.58

[80] Álvaro Villalba, Juan Luis Pérez, David Carrera, Carlos Pedrinaci, and Luca Panziera. servioticy and iserve: A scalable platform for mining the iot. In Elhadi M. Shakshuki, editor, *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), London, UK, June 2-5, 2015*, volume 52 of *Procedia Computer Science*, pages 1022–1027. Elsevier, 2015. doi: 10.1016/j.procs.2015.05.097. URL https://doi.org/10.1016/j.procs.2015.05.097

4

DISTRIBUTION OF DATA PROCESSING UNDER THE FOG PARADIGM

4.1 INTRODUCTION

The interplay between Cloud and Fog computing is crucial for the evolution of IoT, but the reach and specification of such interplay is an open problem. Meanwhile, the advances made in managing hyper-distributed infrastructures involving the Cloud and the network Edge are leading to the convergence of NFV and 5G, supported mainly by ETSI's MANO architecture. This chapter argues that Fog computing will become part of that convergence, and introduces an open and converged architecture based on MANO that offers uniform management of IoT services spanning the continuum from the Cloud to the Edge. More specifically, the first YANG models have been created for fog nodes, for IoT services involving Cloud, network, and/or Fog, and expanded the concept of "orchestrated assurance" to provision carrier-grade service assurance in IoT. The chapter also discusses the application of our model in a flagship pilot in the city of Barcelona.

Several technologies relevant for the expansion of IoT have emerged including Network Function Virtualization (NFV) [2], 5G [45], and Fog computing. In particular, the European Telecommunications Standards Institute (ETSI) has standardized the reference architecture for NFV Management and Orchestration [33], a cornerstone for building, deploying, and managing services in NFV environments. Advances in 5G Radio Access Network (RAN), and in the Multi-access Edge Computing (MEC) group at ETSI [49], are also key for the IoT evolution. MEC proposes a virtualized platform built upon an NFV infrastructure, and is expected to leverage the NFV MANO architecture and APIs. The majority of service providers (SPs) will exploit NFV

infrastructures not only for virtualized RANs and MEC, but also for other services, including enterprise, residential, and Cloud offerings. Thus, the convergence of NFV and some of the key building blocks of future 5G architectures seems unquestionable (Figure 4.1).

Fog computing addresses use cases with requirements far beyond cloud-only solution capabilities. The complementarity between Fog and Cloud has traditionally been seen as a mandatory feature in any Fog platform. In this chapter, a different approach is advocated. Rather than specifying an architecture where Fog and Cloud are complementary by design, we focus on a service management architecture that literally fuses Fog and Cloud. We must start thinking about one computing fabric, managed as a single entity, in a service-centric way. With this approach, an infrastructure composed of Fog nodes, network nodes, and Cloud nodes is exposed to service administrators as a unified resource fabric. Administrators can then define where to instantiate resources according to the service requirements.

Compute nodes in the Cloud or Fog are treated architecturally the same, as the service management platform unifies the life cycle management of services that might require instances running in the Fog, Cloud, or a combination. Distinctive features of a Fog, network, or Cloud node will be captured by their corresponding YANG models [40]. A main advantage of this approach is that different IoT services can coexist and be managed in a uniform way. Consider services where Fog is not required (e.g., sensor communications supported through long-range radio, such as LoRA or NB-IoT [70]) vs. applications where Fog is mandatory (e.g., industrial machines producing data filtered and analyzed by a fog node that is directly connected to the former through a wired interface [9]). System integrators and SPs can leverage our unified infrastructure and uniform service management to provide services to customers in both segments simultaneously.

The requirements to host and manage NFV, MEC, and Fog computing services are undeniably similar. SPs and enterprises embracing NFV will seek to maximize their investments, leveraging their NFV infrastructure and MANO systems to the largest extent possible. It is only a matter of time until Fog computing becomes part of the convergence that we are already witnessing between NFV and 5G/MEC, driven by SPs and enterprise investments (Figure 4.1).

This chapter describes an architectural approach that addresses some of the technical challenges behind the convergence shown in Figure 4.1, with special focus on bridging the gap between Cloud and Fog. We introduce a model-driven and service-centric architecture that is, at the time of writing, perfectly aligned with the OpenFog Consortium (OFC) reference architecture [22].

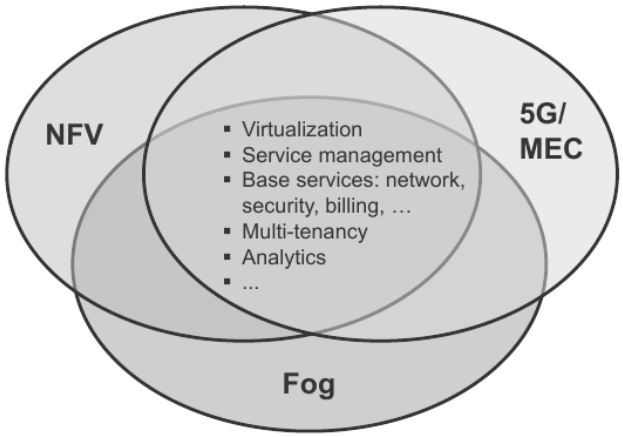


Figure 4.1: Different technologies but with overlapping needs and challenges.

This work has been done in collaboration with the Corporate Technology Group (CTG) at Cisco in order to explore the Fog architecture paradigm to build a Proof-of-Concept (PoC) on Fog Computing.

The PoC addresses the development of the systems required for provisioning, deploying, managing, and maintaining the compute, network, and storage resources needed for running Fog Services.

The PoC ends in a final demo with the Barcelona City Hall to cover different uses cases to address a set of specific functional IoT challenges for cities and demonstrate the strengths of a consolidated service platform for IoT.

The contributions from the candidate to this work relate to its architecture build, data processing, life cycle management and API interfaces extension, mainly:

- Managing the lifecycle of the Fog Nodes.
- Managing the lifecycle of the Fog Virtual Domains (FVDs), i.e., the virtual environments supporting the different Applications running within a Fog Node.
- The APIs and protocols required for managing a Fog System, including the interfaces and protocols: i) within the Backend Platform; ii) within the Fog Nodes; iii) between the Backend Platform and the Fog Nodes; iv) and between the Backend Platform and the External Management Systems.

4.2 TERMINOLOGY

In the literature, a variety of terms are used in various and often inconsistent ways. The following terms are carefully defined for use in this document in order to eliminate any ambiguities.

- Fog Node: The fundamental building block in a Fog Computing system. Fog Nodes provide the compute, storage and networking resources to host Applications. It is worth emphasizing that these are the "physical devices" hosting the Applications.
- Fog Virtual Domain (FVD): An FVD is basically a group of atomic virtual instances that are logically interconnected and that would typically run within a Fog Node, such as a set of interconnected containers or VMs. Note that an FVD can be composed even of a single virtual instance (e.g., just one container or one VM). An FVD belongs to a single administrative domain (i.e., a single tenant), and as such, it must remain isolated from other FVDs inside a Fog Node.
- Fog Service: When a combination of Things, Applications running in the FVDs, and other appli-cations running at upper levels, interact and work in concert, we say that they implement a Fog Service. The ultimate goal of any Fog Computing system is to enable the deployment and execu-tion of Fog Services.
- Fog Region: Fog Regions encompass all the Fog Nodes orchestrated by a (possibly distributed) single instance of the compute, storage and networking components of a Fog Orchestration system.
- Fog Aggregates: A Fog Aggregate is an "overlay" grouping of Fog Nodes within a single Fog Region. Fog Aggregates are optional, and are arbitrarily set up by the Administrator. They are typically used to group Fog Nodes offering similar capabilities to simplify their orchestration. For example, an Administrator may chose to create an Aggregate of Fog Nodes that can access a sensor mesh in less than 50 ms.

4.3 ARCHITECTURE

The objective is to have a single, extensible, and distributed infrastructure that provides the necessary flexibility to address opportunities for current and future urban services technologies in an integrated way. This infrastructure would have nodes in three locations: the cabinets, metropolitan network, and data centers. More importantly, it should enable instantiation and management of urban services and their data in a consolidated way. From an operational standpoint, the goal is to streamline urban services and reduce hardware and service maintenance overheads.

4.3.1 *A Model-Driven and Service-Centric Approach*

The model is based on a two-layer abstraction:

- The separation of the "service intention" (i.e., "what") from the "service instantiation" (i.e., "how")
- The decoupling of the "service instantiation" from the specifics of the devices where the instances will be ultimately deployed - independent of whether they will be instantiated in the Cloud or network, or at the Edge

The left side of Figure 4.2 shows how this abstraction is achieved through utilization of a standardized data modeling language, YANG [40]. The right side shows a small YANG model snippet that is part of a sensor telemetry use case and multi-protocol data aggregation described later. The YANG model shows various parameters related to the tenant, the fog node, and analytics components.

YANG is used for service and device modeling. Models are machine-readable, and can be interpreted and processed by an orchestration system, which is one of the basic components of our NFV MANO implementation. A main role of the orchestration system is to translate the "what" to the "how," and enforce corresponding configurations on specific device models. The translation process is captured by mapping functions depicted on the left side of Figure 4.2, which transform service definitions and input parameters to device configuration parameters. Configurations are enforced through NETCONF interfaces exposed by any device present in our infrastructure (Cloud, network, or Edge). NETCONF is a standard Internet Engineering Task Force (IETF) protocol used to install and update device configurations. The protocol was chosen because of its ubiquitous presence, as it has been largely adopted by SPs and enterprises as part of their service management operations.

The two-layer abstraction is not new. We believe, however, that this is a safe bet toward the convergence shown in Figure 4.1, since this is precisely what many SPs and large enterprises are starting to use when adopting NFV. The novelties introduced in this chapter are:

- The extension of the model to cover fog and IoT. Although the utilization of NETCONF and YANG has traditionally focused on network configuration, these standards are sufficiently generic to be leveraged for any kind of device or service model. We have expanded the reach of NETCONF and YANG to fog nodes.
- The extension of orchestrated assurance to cover IoT services (i.e., service assurance is an intrinsic part of the IoT service definition in YANG).

In our model, everything is reduced to services. Infrastructure services (i.e., those dealing directly with physical resources) become an integral part of modeling and enabling higher-level services. The composition of YANG models for building services is key to breaking

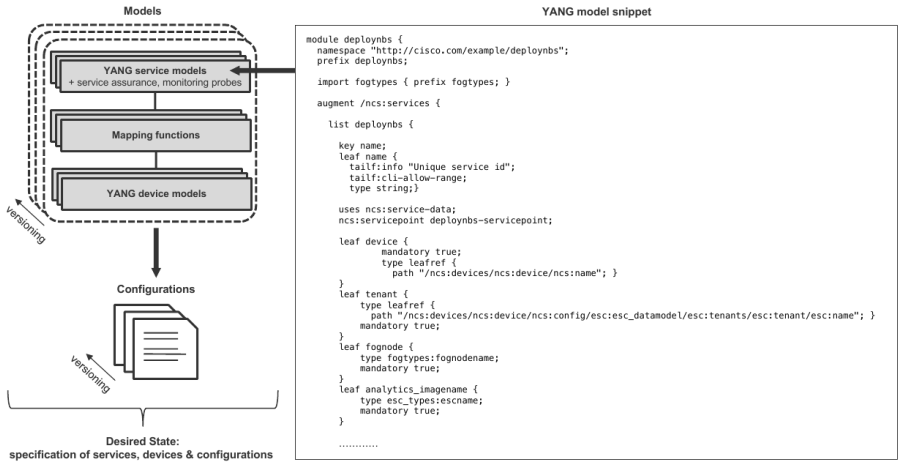


Figure 4.2: YANG is used for both service and device modeling, making devices transparent to service management. Service assurance is supported through distributed monitoring across the infrastructure and feeds a transactional orchestration system, which can deal with any discrepancy between the current state and the desired state of an IoT service.

down the complexity of service modeling and for reusing parts of existing service catalogs. This approach is proven to facilitate life cycle management of large collections of services in NFV environments, and is critical to reduce complexity when designing IoT services, with much more infrastructure heterogeneity beyond the data center.

The workflow for deploying a service starts from a service model definition. Subsequently, the service is instantiated and the configuration is enforced on one or more devices in the infrastructure. As shown in Figure 4.2, YANG models and corresponding device configurations are stored, and can be rolled back to previous models and/or configuration versions should this be necessary.

The designer of a service model can include key performance indicators (KPIs), and compare the "actual state" of the instantiated service to the "desired state" as part of the service assurance. In case of discrepancy between states, service assurance components depicted in Figure 4.3 notify the service managers within the NFV MANO architecture, and orchestration services take action to align the actual state with the desired state.

4.3.2 *Toward Converged Service Management*

Our approach uses the well-known NFV MANO architecture and extends it to other service categories, beyond NFV and network devices. Figure 4.3 shows the main building blocks, split into three categories: the data plane; basic components to support data plane functionality (service assurance, security, and networking); and the management plane based on NFV MANO.

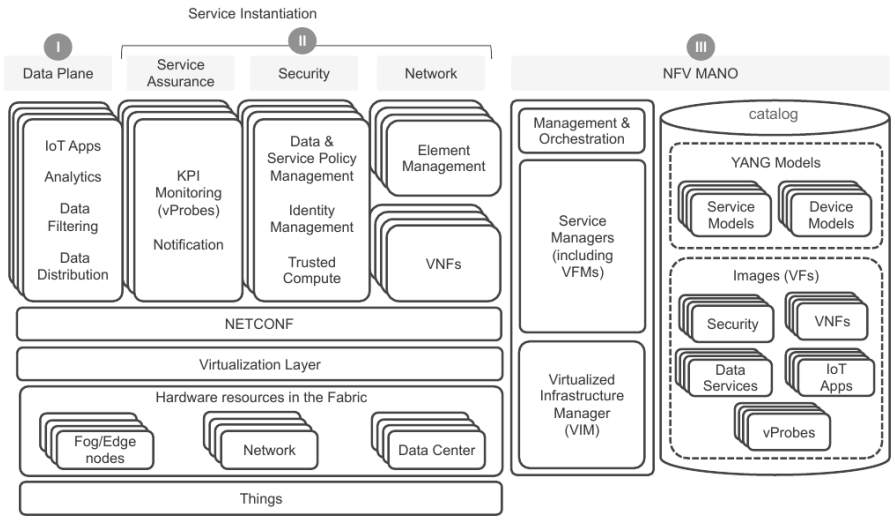


Figure 4.3: ETSI MANO architecture extended to cover service management beyond the traditional NFV and networking domains.

The first category consists of services to manipulate, share, and distribute data to other services over the cloud to edge continuum. The second category provides services to ensure secure and reliable service operation and efficient data delivery. The third category encompasses the usual MANO components extended with new models to cover Fog nodes and IoT services, and IoT-specific features, especially in the areas of security and service assurance. Except for the presence of NETCONF and YANG, the blocks shown in Figure 4.3 are technology-agnostic. NETCONF and YANG are present to emphasize the need for the adoption of standardized and broadly accepted interfaces and data modeling languages.

The main components are described in more detail below.

- Data Plane: Refers to the transport and delivery of the broadband traffic that are flowing. Includes data distribution and data sharing services. The data and service policy management module present in the security block (II) allows administrators to share data between tenants in a controlled and secure way. This enables sharing of data between services on multiple Fog nodes, and also across Fog and Cloud, while adhering to policies defined in the data and service policy management module. These policies could be leveraged to build so-called data and resource pricing models [48] as an incentive to optimize resource usage in multi-tenant environments.

  The setup should offer multi-tenancy, and enable streaming and/or historian analytics depending on requirements.

- Service Assurance: Services are linked to a certain quality of service, specified by KPIs. An effective technique starting to be used in NFV scenarios is to monitor the KPIs through a set of

virtual probes (vProbes), instantiated at the points where relevant parameters need to be monitored, and usually deployed in a distributed way. A combination of passive and active vProbes can efficiently detect violations to KPIs directly at the problem source.

This technique has proven simpler and more accurate than traditional approaches, which are often based on gathering information from multiple sources, including measurement tools, logs, monitoring systems, and so on, for root cause analysis. In our platform, KPI violation is notified to the service owner, and orchestrator or service manager, to resolve the discrepancy between observed and desired service state. Service assurance forms an integral part of the service definition and composition developed in YANG (captured through service models). The role of vProbes, their locations, and actions that need to be taken upon KPI violations are specified in the service model. Service assurance covers both the infrastructure and the services that multiple tenants will deploy on top of it.

- Security: An integral part of the architecture, since the extension of NFV MANO to cover Fog and IoT substantially increases the attack surface. Security elements are categorized into the following three groups.

  - *Network-Based Security and Role-Based Access Control:* Provided through specific virtual network functions (VNFs), such as firewalls, intrusion detection applications, and so on. Many of these will be instantiated in fog nodes, and service designers decide whether a security VNF is instantiated to protect an entire node (e.g., a Fog node), a specific tenant execution environment (TEE) within a node, or a pool of TEEs instantiated in a single chassis (where one or more Fog nodes can reside) or spanning across several of them. This category also covers mechanisms for controlling which services can access what data and when, as well as which users can access what services and resources and when. Mechanisms for authentication and authorization are essential to control data sharing policies and grant access to specific resources for each tenant. Different tenants usually deploy different services, and require different KPIs to be monitored. In the case of Barcelona, data associated with service assurance of different tenants was stored in a geo-distributed historian database. Enforcement of access control on the database ensured separation of information between tenants. Internal communication between components of the platform were also secured using encryption.

- *Host-Based Security:* Includes aspects such as trusted compute based on the trusted platform module (TPM), operating system hardening, disk encryption, vulnerability management and patching policies, security information and event management (SIEM), enforcing isolation among TEEs, and so on. The Barcelona pilot covered the majority of these aspects, with strong focus on securing fog nodes.

- *Fog-Based Security:* Security the system can provide to help protect "things" connected to Fog nodes, and protect the fabric and its services from malicious things located at the network edge. The IETF has recently proposed the manufacturer usage description (MUD) specifications [53] as a first step toward a standardized and secure way of onboarding, and connecting, simple "things" to an IoT system. Fog can play an enabling role for MUD, to mediate and automate the process of device onboarding, and to enforce security policies ensuring such devices can only establish communications subject to their intended use.

- Network: This covers the core networking VNFs and ancillary systems, such as virtualized switches, routers, DHCP servers, load balancers, WAN optimizers, and so on.

- NFV MANO: Unlike traditional NFV deployments, where services are instantiated in environments with homogeneous IT and network infrastructures, for many hyper-distributed IoT environments, heterogeneity of devices and communications is more the rule than the exception. Capturing this heterogeneity in simple, standard, and machine-readable ways is essential. YANG models provide this, since not only can network elements be modeled but also fog nodes, elementary things using MUD specifications [53], as well as IoT services to be deployed. The YANG model snippet in Figure 4.2 was used in Barcelona, and was part of the catalog of services and device models shown on the right of Figure 4.3.

  The NFV MANO block in Figure 4.3 is based on ETSI's three-tier model:

  - Management and orchestration.
  - Service managers, supporting multiple vendors.
  - The virtualized infrastructure manager (VIM) Traditional VNFs in ETSI's MANO terminology.

  Traditional VNFs in ETSI's MANO terminology correspond to a subset of virtual functions (VFs) managed by our architecture, with many of our VFs containing IoT-related functions rather than only network functions. A certain level of atomic behavior when updating many services at the edge is a necessity. The

MANO system achieves this through transactional operations across services involved. This is similar to a two-phase commit across multiple databases, ensuring physical devices associated to these services continue to function properly, and the system as a whole stays in a consistent state.

Services are deployed by combining the YANG models, associated images for the VFs to be instantiated and configurations of networks, message brokers and data flows, security policies, databases, and so on supporting the service. Either a user will specifically push a new service to a set of edge, network, or data center nodes, or, depending on the KPIs and overall system state, MANO will determine the best possible location for services to be deployed. Because all fabric hardware resources have NETCONF interfaces and are described through appropriate YANG device models, from a deployment perspective, there is no distinction between edge, network, or data center nodes.

4.3.2.1 *Technologies for Implementing the Architecture*

While previous paragraphs describe the main architectural components, Table 4.1 shows various technologies that can be leveraged to implement them, including ones used in the Barcelona pilot. YANG models of Fog nodes or service components can be implemented by various hardware and software vendors (or developed by the open source community). These models can become part of our implementation. Indeed, several YANG models and service templates could become part of the OFC interoperability trials [22], and, together with the extended MANO architecture presented in this chapter, form part of a reference framework for open implementations.

4.4 MOTIVATION AND PILOT IMPLEMENTATION IN THE CITY OF BARCELONA

Barcelona realized that the more than 3000 street cabinets deployed in the city form a natural infrastructure to build out their smart city vision. Their goal is to have a single, extensible, and distributed platform from Edge to Cloud to address opportunities that current and future technologies for urban services will bring in an integrated way. The incentives behind this approach are described in detail in [86], but one of the aims is to reduce solution silos and the cost of operating different solutions in the city. While [86] addresses multiple use cases where Fog is mandatory, this section delves into the orchestration needs, and outlines the automation and uniform life cycle management of two use cases spanning the cloud to Edge continuum (Figure 4.4). These use cases were recently demonstrated in Barcelona, and extend those described in [86].

Component	Technology Ecosystem
Analytics	**Cisco ParStream**, Cisco Edge and Fog Fabric (EFF), Apache Storm, GE Predix, SAP, etc.
Data filtering and normalization	**Various processes for anomaly detection including Kalman filters, NearbySensor Agent for data normalization, etc.**
Data distribution	**RabbitMQ**, Cisco EFF, Apache ActiveMQ, DDS, etc.
vProves (service assurance)	**netrounds probes, NearbySensor Assurance, etc.**
Data and service policy management	**Specifically implemented during the project.**
Identity management	**LDAP and distributed replicas**, Cisco ISE, Active Directory, etc.
Trusted compute	**TPM/TXT, OSSIM, Open Attestation, LUKS, SELinux, AppArmor, QEMU, and secured configuration files.**
VNFs	**Cisco CSR1kv, Cisco Firewalls**, Cisco ESR, Palo Alto Firewalls, load balancers, etc.
Management and orchestration	**Cisco tail-f NSO**, Puppet, Chef, etc.
Service managers (VFMs)	**Cisco Elastic Services Controller (ESC)**, Ciena Blue planet, Brocade, etc.
Virtualized infrastructure manager (VIM)	**OpenStack**, vSphere, etc.
Fog hardware	**Cisco IOx devices, Nebbiolo Technologies, NearbySensor Box, ADLink, Darveen, etc.**
YANG models	**Various models for services and devices: data sharing, analytics, NearbySensor VFs, Fog nodes, etc.**

Table 4.1: Potential technologies for implementing the main components depicted in Figure 4.3. In bold are the ones used in Barcelona, and we also list other alternatives where applicable.

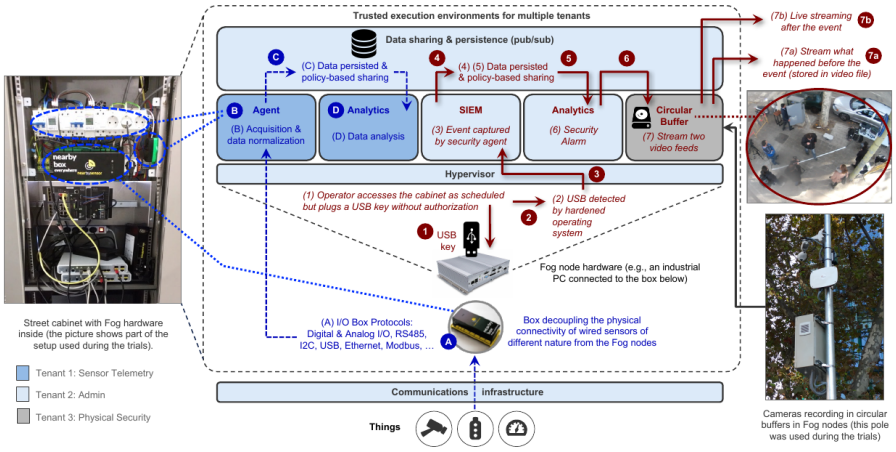


Figure 4.4: Fog architecture deployment.

4.4.1 *Dealing with Scale and Management Complexity*

Fog nodes were housed inside cabinets, such as the one shown on the eft of Figure 4.4. For resiliency, some services require more than one Fog node per cabinet. A large fraction of cabinets will host instances of the same service, so managing the life cycle of a single service across the city may involve the configuration of thousands of Fog nodes.

The development of an IoT service usually entails the integration of multiple technologies supplied by a partner ecosystem, including sensors, application-specific gateways, Fog and network nodes, data brokers, security, and so on. Thus, managing the life cycle of an IoT service (i.e., onboarding devices, performing day zero configurations, as well as managing the state and configurations after the initial deployment) can become quite complex - a challenge faced in Barcelona, and other cities and industries.

The NFV and 5G communities have addressed similar challenges, both heavily betting on the ETSI MANO architecture. We argue that this architecture will not only facilitate the convergence of NFV, 5G/MEC, and Fog, but will also offer the automation means to deal with the scale and complexity posed by IoT. The goal is to hide underlying complexity from administrators, and turn IoT service management into simple and intuitive operations. The success of platforms such as Amazon's AWS Lambda, Google's search engine, or legacy technologies like TV is largely due to the way they manage scale, and the way they have abstracted the underlying complexity from end users. The pilot conducted in Barcelona followed the same principles.

4.4.2 *Setup in Barcelona*

Figure 4 offers a schematic view of a setup used during the Barcelona pilot. The left side shows a cabinet interior with several elements:

- A power distribution board with different monitoring elements and circuit breakers.
- A box that enables decoupling and aggregating the physical connectivity of different families of wired sensors, simplifying the I/O requirements of the fog nodes.
- The fog nodes themselves.
- Others

The central part of the figure provides a logical representation of the setup, and shows data flows for two different use cases demonstrated in Barcelona. The bottom shows a set of "things" (e.g., sensors and control and actuation elements), which can be located both within and outside cabinets.

We used Fog nodes supplied by different vendors (Table 4.1). The example in the figure shows an industrial PC, connected through Ethernet to the NearbySensor box. The top of the figure illustrates the hypervisor as well as several TEEs, which belong to three different tenants running in the Fog node. The right side shows part of the external setup for one of the use cases, including a pole, a camera, and a snapshot of one of the videos captured by the latter.

We proceed to describe two use cases depicted in Figure 4.4, with emphasis on automation enabling the data flows illustrated therein.

4.4.3 *Use Case 1: Sensor Telemetry through Street Cabinets*

From a data plane standpoint, this use case is depicted as sequence A-D in Figure 4.4. Step A shows how specialized hardware at the network edge can help aggregate and simplify communication with different types of sensors (e.g., for monitoring temperature, power, and access), as well as a number of controllers, such as circuit breakers and uninterruptible power supplies, using various protocols and interfaces.

Data collected through the box in A is sent to an agent (B), which normalizes the data. This agent is part of a TEE that belongs to the city department in charge of monitoring the cabinets and the environment (tenant 1 in Figure 4.4), and can run in a Docker container or a VM depending on security and performance requirements. Data processed by the agent can be maintained and shared in a policy-based and secure way with other processes running either on the fog node, on other fog nodes, or in the Cloud (C). The data sharing and persistence block on the top belongs to the administrative tenant in charge of managing common services across tenants (tenant 2), such as data sharing policies, system-level analytics, and the security of the Fog node itself. Step D shows tenant 1 subscribed to different data topics, enabling the gathering and examination of data from different sources, and triggering actionable decisions based on the result of the analysis. Applications (and their YANG service models) running in B and D can be supplied by different providers.

In the example, the process in D analyzes - among other things - power consumption obtained from monitors connected to the box in A, and estimates upcoming values based on a Kalman filter (Figure 4.5). The goal of the analysis is three-fold:

- Estimate and control the power consumed to prevent spikes in energy use from multiple devices.
- Dynamically manage which devices remain operational in case of a power outage.
- Control the service level agreement (SLA) with the energy supplier. Since the process- es run locally in the fog node, this anal-

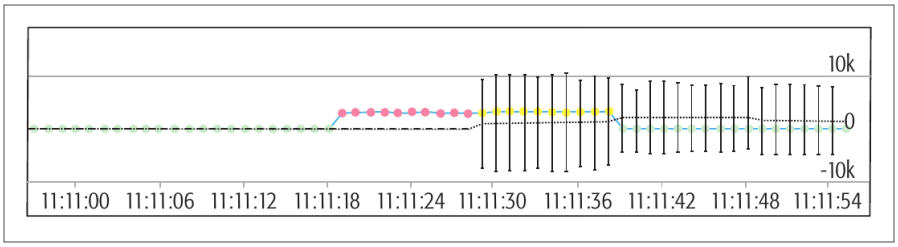


Figure 4.5: Power consumption data (in Watts) associated with sequence A-D depicted in Figure 4.4. The dots represent the observed values B-C; the dotted curve represents estimated values and uncertainty using a Kalman filter (D).

ysis and control will remain operative even if the node loses backhaul connectivity to the cloud.

The deployment of services supporting the use cases depicted at the center of Figure 4.4 was entirely automated, and managed using the architecture shown in Figure 4.3. Configuration of Fog nodes was performed as follows:

- Zero-touch provisioning including full installation of operating system, initial configurations, security, and so on.
- Deploying and configuring initial function packs, such as data sharing and persistence elements, a set of vProbes for service assurance, and more.
- The tenant's TEEs, the security configurations enforcing segmentation and isolation between tenants, including their corresponding networks.
- Configuration of message brokers enabling the data workflows (A-D) shown in the figure.

All stages required for deploying and configuring the IoT services shown in Figure 4.4 were managed with a few clicks. More importantly, instantiations can be done in an individual cabinet or thousands of them across the city once the service models and the corresponding images are available from the catalog illustrated on the right side of Figure 4.3. This approach reduces the operating expenditure for managing a smart city infrastructure considerably.

4.4.4 *Use Case 2: Physical Security of Fog Nodes*

Physical security of cabinets and the devices inside is of utmost importance for the city, so we implemented several security layers. This section describes how unauthorized USB access to a fog node is detected and recorded on video (cf. the right side of Figure 4.4).

Cameras record continuously and send video to associated Fog nodes, where they are stored in circular buffers. Only when an event occurs, such as inserting a USB key, does the Fog-based system trigger two video streams: what happened before the event (stored in the circular buffer and what happens right after the event in real time. The reasons for not streaming continuously to the Cloud, but deploying services at the Edge using fog, are cost, privacy, and data storage overheads [86]. Actions taking place after the USB stick is plugged in (step 1 in Figure 4.4) until the video is stored and made available correspond with steps 2-7 in Figure 4.4: 2) the USB is detected, generating an event; 3) the event is captured by a SIEM agent; 4) and 5) reporting the event through the communication bus; 6) analysis of the event and triggering the appropriate response; and 7) streaming the videos to one or more predefined locations (e.g., in the cloud). Communications between the use case components occur transparently, and there is no semantic separation between Fog and Cloud deployed components.

An important aspect is the multi-tenant nature of the converged Cloud/Network/Fog platform, as services supporting this use case were deployed on the same Fog node used before. As in the previous use case, not all services and hardware needed for implementing the use case were managed by the same city department. The IT department manages the services running in the cabinets, while the cameras and their functionality are managed by another department. We leveraged common functions offered by the platform, including the data sharing service, various security and service assurance functions, and so on.

This use case demonstrates that, an IoT service is typically composed of multiple services. Services can be managed by different tenants or securely shared among multiple tenants (cf. the data sharing service in Figure 4.4). Some services may be deployed and associated solely with an IoT service and tenant (e.g., the circular buffer service in Figure 4.4). Besides service composition and re-usability, the aim is to turn the deployment and management of IoT services into almost trivial tasks, which can be operated through a set of intuitive actions (performing a few clicks on a dashboard). All these aspects are at the heart of our converged architecture.

4.5 RELATED WORK

The authors in [44] discuss how some of the MANO concepts can be exploited to deliver end-to-end network services using a description-based approach over a set of distributed resources. This work has similarities with ours, although our focus is mainly IoT and Fog, whereas [44] is centered on the orchestration and deployment of network services. Note that the work discussed in this chapter sits at

the intersection of NFV, 5G/MEC, and IoT and Fog, and extends the concept beyond the data center and networking to fuse Cloud and Fog. While MEC focuses mainly on the edge of the network [49], our approach covers the continuum from Edge to Cloud.

Service configuration and life cycle management are important aspects of our platform. Several products such as Puppet, Chef, Ansible, and Salt provide configuration management and help automate deployment of services. Other products like Terraform, CloudFormation, and OpenStack provide infrastructure life cycle management capabilities (e.g., for day 0 configuration), which can be combined with tools like Puppet or Chef (for day 1 onward). However, all these products mainly target IT infrastructures. In IoT, the fundamental requirements in terms of connectivity (I/O interfaces), security, applications, and data management are significantly different from those that rule the life cycle management of IT servers. The compute resources available in a Fog environment are typically much more heterogeneous, and the criticality of some applications requires special treatment.

This heterogeneity, combined with the criticality of some of the services connected to physical devices, creates new requirements for service life cycle management that go far beyond what state-of-the-art tools in the IT space currently offer. Our approach takes into account this heterogeneity natively.

Finally, many of the members of the OFC are already offering fog products. This is the case of Foghorn, Nebbiolo, and Cisco, just to name a few [22]. At the time of writing, none of the products available in the marketplace are focused on a converged NFV, 5G/MEC, Fog, and Cloud paradigm, with emphasis on exposing the infrastructure as a single and unified computing fabric.

4.6 SUMMARY

This chapter describes an architecture that addresses some of the central challenges behind the convergence of NFV, 5G/MEC, IoT, and Fog. By using a two-layer abstraction model, along with IoT-specific modules enriching the NFV MANO architecture, we introduce a promising paradigm to fuse Cloud, network, and Fog, and apply this to a project in the city of Barcelona. For now, we focus only on a small number of use cases. We expect that once we start expanding this model to different domains and cities, more end-user services will be developed, enabling the reutilization of service models and associated service catalogs.

The work described in this chapter is based on the following main publication:

[86] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, and Albert Olive. A

new era for cities with fog computing. *IEEE Internet Computing*, 21 (2):54–67, 2017. doi: 10.1109/MIC.2017.25. URL https://doi.org/10.1109/MIC.2017.25

[79] Frank van Lingen, Marcelo Yannuzzi, Anuj Jain, Rik Irons-Mclean, Oriol Lluch Parellada, David Carrera, Juan Luis Pérez, Alberto Gutierrez, Diego Montero, Josep Marti, Ricard Maso, and Juan Pedro Rodriguez. The unavoidable convergence of nfv, 5g, and fog: A model-driven approach to bridge cloud and edge. *IEEE Communications Magazine*, 55(8):28–35, 2017

5

DISTRIBUTION AND SCALABILITY OF IOT OPERATIONS WITH MOVING DATA SOURCES

5.1 INTRODUCTION

Several technologies relevant to the expansion of the Internet of Things (IoT) have emerged in the last years, including network functions virtualization (NFV), fifth generation (5G) wireless systems, and Fog computing. The combination of these technologies opens a new range of potential applications in the context of Smart Cities. There is a fast growth in the number of projects planning to deliver new services to citizens, based on the deployment of a large number of Fog nodes near the edge, in the streets of modern cities, bridging the gap between devices and Cloud-based services. Fog nodes can host lightweight services on near real-time, like for instance the collection and processing of streams of data. This technology is a foundational enabler for the future development of advanced services such as for instance traffic monitoring and planning through the combination of street sensors data (e.g. vehicle tracking, air quality measurements) and meteorological information. Although Fog nodes offer a constrained computing capacity compared to their Cloud counterparts, they still have capabilities to process data in near real-time to provide localized service to users, minimizing the communication requirements with the Cloud, or ensuring application resilience against back-haul connectivity outages between the Fog and Cloud layers.

Modern cities demand new approaches to deliver localized services to their citizens, and at the same time, network operators look for new advanced services that can take advantage of the new hyperconnected society that is expected for the coming 5G era. The incredibly high bandwidth that 5G networks will offer to their users will

restrict the possibility to define new services that run only in centralized Cloud-based locations. Therefore, the development of decentralized architectures that leverage the Fog computing paradigm (computing between the edge and the Cloud) is a mandatory requirement for an efficient deployment of 5G technologies over the next couple of years. In this context of highly connected cities with distributed Fog-based computational capabilities, applications will require a superior ability to adapt to the continuous changes that occur within the dynamics of a modern city: the only way to provide the required flexibility will be through the use of advanced Artificial Intelligence techniques that help systems *learn* and model the behavior of crowds in near real-time. It is only under these conditions, with the combination of 5G networks, the Fog computing paradigm and the exploitation of AI techniques, that it will be possible to develop the complex services that cities demand.

In this chapter, a distributed architecture for a traffic modeling and prediction service is presented, designed for a city-wide scenario based on the Fog computing paradigm. In this context, a set of advanced antennas are assumed (e.g. 5G stations [52] enabled with Fog computing capabilities, acting as a Fog node) are distributed across the city, and they are used to receive telemetry and location data as generated by vehicles. Each vehicle sends data to the nearest antenna and its associated Fog node. Therefore, data is collected and locally processed in Fog nodes (either located at the Edge or in-between the Edge and the Cloud as intermediate nodes), and then forwarded to a central Cloud location for further analysis as well as data warehousing purposes. The proposed architecture combines a real-time data distribution algorithm with enhanced resilience against backhaul connectivity issues, and a traffic modeling technique based on the use of Conditional Restricted Boltzmann Machines (CRBM) to learn traffic patterns. In combination, these two techniques provide resilient and completely decentralized city-wide traffic forecasting capabilities.

The proposed architecture is validated using real traffic logs from one week of Floating Car Data (FCD) in the city of Barcelona, provided by one of the largest road-assistance companies in Spain, comprising thousands of vehicles from their fleet only in the city of Barcelona. The dataset (further described in Section 5.5.2) comprises data collected over one week between 10/27/2014 and 11/01/2014 across the Barcelona metropolitan area. Figure 5.1 shows a heat-map of the vehicle tracking data, comprising over 890,000 data samples and a fleet of more than 100 cars moving simultaneously around the city at some times.

Using this FCD dataset, a simulation using the provided FCD across several conditions are done, from scenarios in which no connectivity failures occurred between the Fog nodes and the Cloud, to scenarios

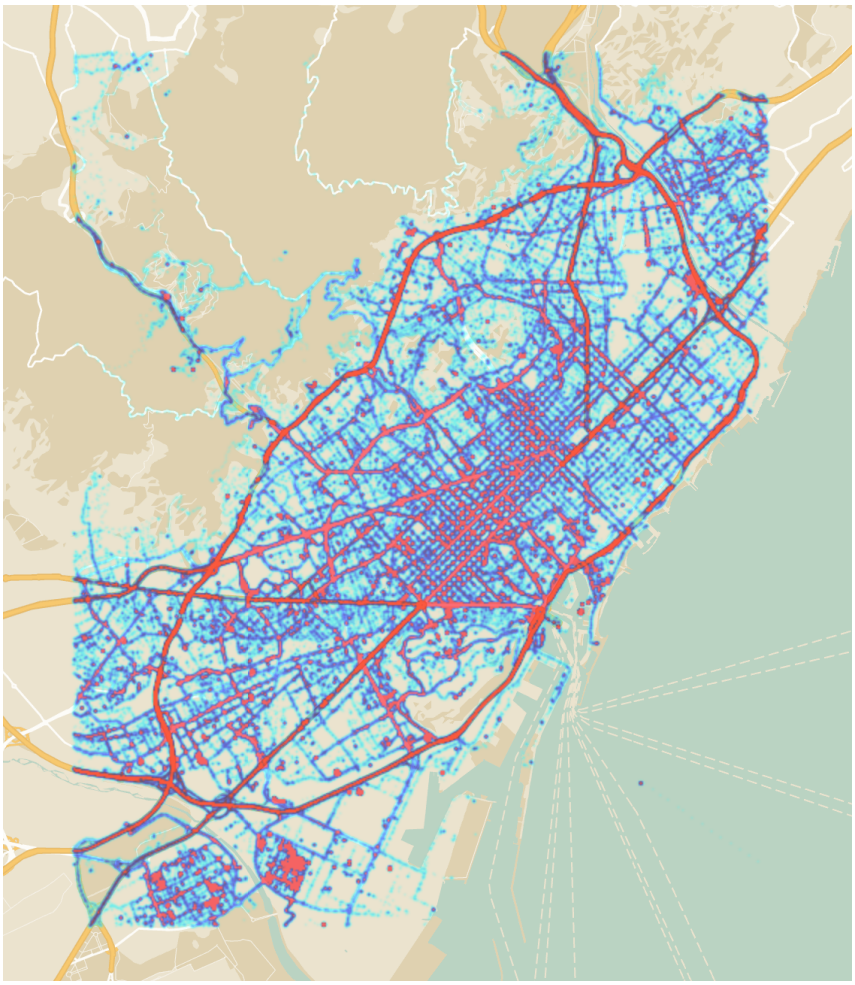


Figure 5.1: Barcelona metropolitan area map, combined with a heat-map overlay of the FCD dataset used for the simulations presented in this chapter. The dataset contains more than 890,000 data samples of road-assistance cars moving around the city.

with long and frequent connectivity outage periods. For each one of those scenarios, the resilience and accuracy of the data distribution algorithm and the learning methods have been analyzed.

While current frameworks dealing with FCD analytics focus on how to distribute load towards anomaly detection, modeling and trend prediction on Cloud infrastructures and leveraging Map-Reduce mechanisms to handle traffic data, in this work the focus is on 1) the scenario where analytics can be partial or completely performed on the Edge instead of on the Cloud; and 2) the proper transmission of data between Fog nodes on the Edge and the Cloud towards delivering data to be aggregated or learned models to be used. The current case of use targets city-wide traffic data, but Edge-Cloud architectures can be used for enhancing Smart City applications, like power monitoring and control of elements in public spaces, connectivity on demand from smart phones towards public services, or sensor

data recopilation from smart phones towards retrieving environmental data [86].

Experiments show that the here presented architecture for data distribution running in the Fog nodes is resilient to back-haul connectivity issues, and it is able to deliver data to the Cloud location even in presence of severe connectivity problems. Additionally, the proposed traffic modeling and forecasting method based on CBRMs, not only is able to predict telemetry features at short terms but also exhibits better behavior when modeling local data at Fog nodes instead of a centralized model in the Cloud, useful when connectivity issues force data to be delivered out of order to the Cloud, providing an extra degree of autonomy to Fog nodes.

In summary, the three major contributions of this chapter are:

- Data Distribution algorithm for FCD collection in city-wide Fog deployments. The algorithm is designed to be resilient to back-haul connectivity issues, to avoid data to be lost under connectivity outage periods, and to favor distributed data modeling in the Fog. The chapter also provides an analysis of the behavior of the algorithm under different scenarios of lost connectivity.
- A distributed traffic learning and forecasting model, particularly designed for Fog deployments in the city, in which data collected by the data distribution algorithm is fed into a distributed set of Conditional Restricted Boltzmann Machines (CRBM). The neural networks learn traffic patterns across the city and can be leveraged to forecast future traffic conditions. The distributed approach is superior to a Cloud-centralized schema in terms of resilience against network connectivity outages.
- Validation of the two previous elements through the simulation of different network stability conditions, using as a source real FCD data collected in Barcelona for one week period in 2014.

The chapter is structured as follows: Section 5.2 introduces the background on distributed architectures and IoT management. Section 5.3 presents the proposed solution towards the current problem. Section 5.4 describes in detail the components of the presented approach. Section 5.5 shows the different evaluation experiments for the current proposal. Section 5.6 provides relevant related work. Finally, Section 5.7 provides a summary of the chapter.

5.2 EDGE ANALYTICS AND FORECASTING WITH CRBMS

5.2.1 *Analytics on the Edge versus Cloud Analytics*

An aspect to consider when computing analytics on the Fog is whether the analytics are performed in the Edge, in the Cloud, or in an intermediate level. Such analytics can be focused on aggregating data (e.g.

counting and data-stream analytics), or in characterizing data (e.g. modeling and machine learning analytics for prediction).

Computing on the Edge often implies Fog nodes to keep a critical mass of data or partial data aggregations, depending on the complexity of the analytics to be performed, plus enough computing power to process data. Otherwise, modeling on the Cloud requires moving data and local aggregations up, then returning the models and analytics if to be used in the Edge, depending on communications but enabling more complex analytics. In those scenarios that data cannot be directly processed on the Edge, but connectivity to the Cloud needs to be economized, intermediate Fog nodes can be enabled in between Edge and Cloud, to collect data and produce aggregated analytics and models. The Fog computing paradigm allows the design of a hierarchy of nodes from Edge to Cloud, placing those analytics and modeling processes in the most suitable level according to connectivity and computing power. Figure 5.2 shows the Fog computing paradigm on aggregation and data processing levels between the Edge and the Cloud.

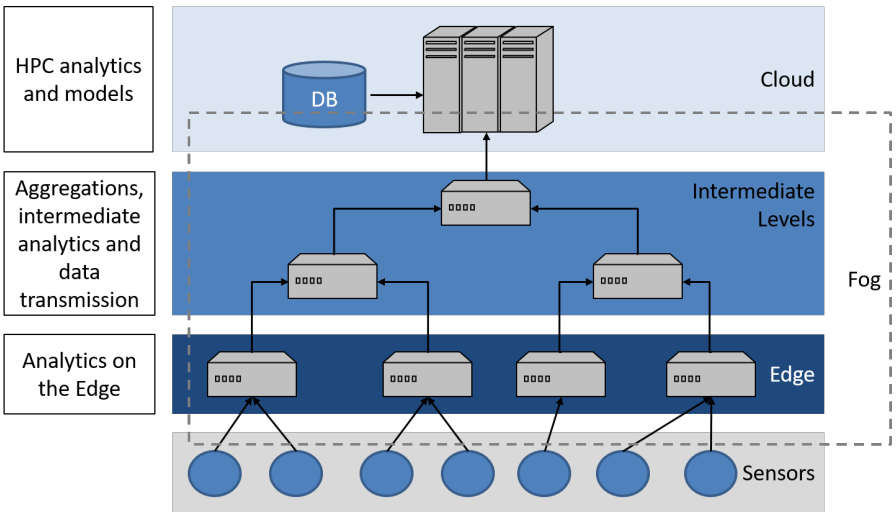


Figure 5.2: Fog computing and different levels between Edge and Cloud

Furthermore, in scenarios with low amount of data per node, machine learning methods may become under-trained if a critical amount of useful examples is not met, or reaching this amount may take too much time. Then, collecting data in upper levels and the Cloud, could provide much faster a higher amount and more diverse data, with all nodes cooperating to provide a valid training dataset. This would be the case for a model trained from all collected data, or when examples from one node can complement examples from another to avoid model over-fitting situations. On the other hand, scenarios with enough data per node can venture to create machine learning models

locally, becoming independent from other nodes or the connection with the Cloud.

Finally, another aspect to consider when computing analytics in the Edge or in the Cloud, is the frequency of updates. While on off-line machine learning methodologies, a training dataset is compiled once to produce a model that is distributed once, on-line machine learning methods require to define update policies indicating the periodicity of model updates and replacements. The advantages of off-line learning is that once the model is created (and distributed if applies), no further operation is required, but if data changes over time such models become outdated. The advantages of on-line learning is that models can be created from few or no data, the updated as data keeps coming, but the training process must be periodically repeated.

5.2.2 *Conditional Restricted Boltzmann Machines*

In this work Conditional Restricted Boltzmann Machines (CRBM) have been used for modeling and forecasting, a Machine Learning technique proposed by G. Taylor et al [74]. CRBMs are an extension of a Restricted Boltzmann Machine, specially (but not only) designed to handle sequential data. CRBM has been chosen among other time series methods because they provide 1) a representative non-simplistic aggregated analytics involving data processing, not as simple as e.g. data-stream sketches and not as complex as e.g. convolutional neural networks or other machine learning ensembles; 2) a method to produce on-line exportable and updatable models, as the set of matrices composing a CRBM model can be easily transported and re-trained; and 3) the capability for long term forecasting, that might be useful in scenarios where the Cloud requires to estimate the status on the edge but communications are interrupted.

The CRBM modeling presented here uses a Gaussian Bernoulli RBM (GB-RBM) to model the static frames of the input time series. While standard RBMs model only binary data, GB-RBMs are used to handle real and integral valued components. The GB-RBM is an Energy-Based Model with Gaussian visible variables (inputs) and hidden Bernoulli variables (featurized representation). Variables in this type of models are also called "units" or "neurons". The GB-RBM configuration was used as in Taylor's [74] and Salakhutdinov's [72] approaches.

The CRBM models used are essentially a GB-RBM with extra inputs to model temporal dependencies. To be specific, to train a CRBM forecasting model, a history window was kept then feed the model with each current input plus n previous steps. The training process is done through a Contrastive Gradient Descend [50] iterative process, where each sample (with its n previous samples) is seen by the CRBM. Through this, the CRBM learns the relation between input and history,

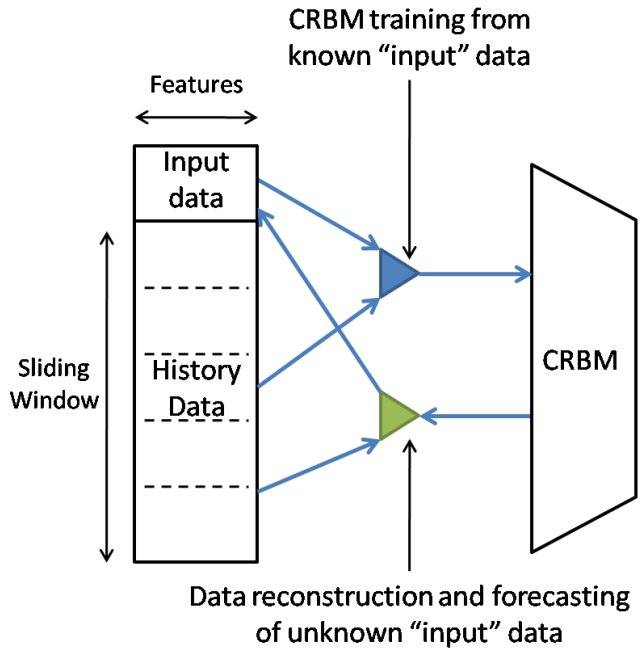


Figure 5.3: Schema of CRBM training and prediction

allowing input prediction at k future steps, through a Gibbs sampling. The advantage of CRBMs over other methods is that it directly handles multi-dimensional input vectors, also allowing constant updates and retraining. Figure 5.3 shows the basic CRBM schema.

CRBMs are used in other fields for characterizing and predicting time series, like in Taylor's work [75] for human motion modeling or in Cai's work [43] for financial data modeling, providing enough experimental support to consider CRBMs a proved stochastic method for time series forecasting, with dimensionality reduction capabilities and able to be updated on-line.

5.3 SYSTEM ARCHITECTURE

The solution proposed in this paper is based on the Fog computing paradigm, combining a data distribution algorithm oriented towards data collection resilient to back-haul connectivity issues, and a traffic modeling approach providing distributed traffic forecasting capabilities, based on the aforementioned Conditional Restricted Boltzmann Machines (CRBM). Figure 5.4 shows the schematics of the proposed architecture. In this architecture, the Fog nodes (including nodes in the Edge) implement the control plane, providing the implementation for data collection algorithms and analytics engines (including machine learning modules, here the CRBMs), also the mechanism for pushing data and models towards upper levels and the Cloud.

The Cloud level, containing the end-points for the Fog node hierarchy, provides the Data Store with scalability and distribution capabilities, also fault-tolerance mechanisms; the System State monitoring mechanisms, continuously providing status information for all the architecture components; and analytics requiring global data and high performance computing resources.

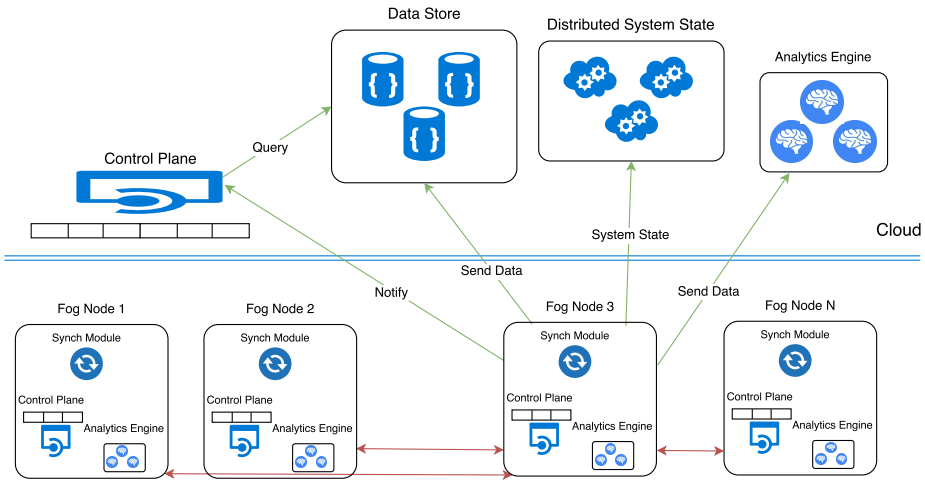


Figure 5.4: Schema of the proposed architecture at Fog level and Cloud level

5.3.1 *Traffic modeling in the Edge vs. Cloud: tradeoffs*

Given the proposed scenario, two approaches when modeling traffic are considered: performing the analytics on the Edge or in the Cloud. Computing models on the Cloud requires collecting all data from Fog nodes, being constrained by possible transmission failures, while computing models on the Edge requires some computing power to perform the analytics. As shown in Figure 5.5, modeling requires data susceptible of transmission disruption, or low-powered devices powerful enough, depending on where modeling is taking place.

Location of models also depends on the network capacity and intended use for analytics. One of the purposes of computing on the Edge is to save data transmission by pushing only aggregated (modeled) information to the Cloud. Infrastructure architects must consider the Fog nodes capacity to produce such aggregations, in front of network bandwidth and availability, so more aggregated data towards low transmission volume requires higher computing power, and vice-versa. Further, scenarios requiring analytic models on the Edge, and analytics being aggregated on the Cloud (individualized or generalist models requiring HPC resources), may require models to be pushed back from Cloud to the Fog nodes. Then, those archi-

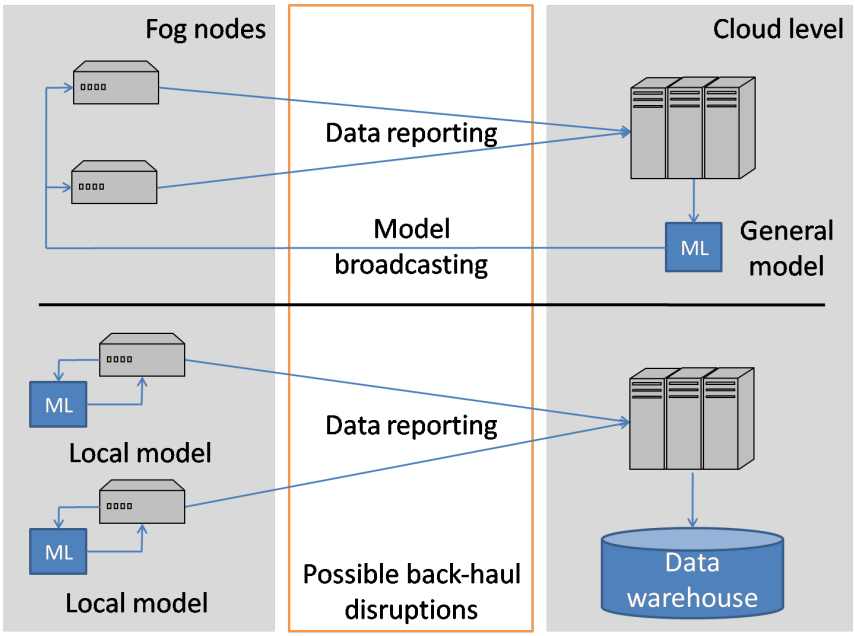


Figure 5.5: General model in the Cloud vs localized models on the Edge

tectures depending on data/model transmissions along time must be aware of back-haul disruption problems.

5.3.2 *Modeling Architecture*

A model is created by collecting data (from all nodes in the general model, or from a single node on individual models), training or updating a CRBM able to forecast $t+1$ telemetry data from $t \ldots t-d$ history, where t = time, d = history memory. As the Cloud collects data from all nodes, a general model can be created from all collected data (here location features for each data value are useful), or individual models can be created for each node to specialize them. In Fog nodes, local data is used to create a specialized model for that node. The CRBM learns and forecasts the following features # *cars* (volume of traffic during that time interval), and *average speed*, from previous traffic volume and average speed, also includes information about the location of the Fog node (latitude and longitude). Further, information about the position of the FCD source is also available, towards discriminating data from specific traffic areas or streets when performing aggregations; while at this time a specific distribution of antennas are considered to discriminate traffic areas (heavy traffic streets, residential areas, etc), when antennas are arbitrarily distributed, data source position is required to perform proper discrimination towards aggregations. Fog node location becomes useful when creating a global model, being able to forecast information from any location.

The CRBM policy on data arrivals require that data is sorted by time-stamp, as usually models learning from time series require. This makes critical for intermediate Fog nodes and the Cloud to receive data in order towards proper aggregation and learning. For empty time-stamp gaps (when no data arrives for a certain time), data is complemented with zeros: entries with the corresponding time-stamp, the last observed latitude and longitude (considering that Fog nodes are fixed in place), and 0 at each other feature. Empty gaps can respond to "no data emitted" or "data didn't reach the cloud". At this time, from the CRBM at the Cloud point of view, both situations are indistinguishable not knowing if silence is due to a connection failure or no data to be received. So the standard policy is to fill "not available data" with the default values, as cannot any other values be assumed. This treated new batch of examples is then split into mini-batches to be fed to the CRBM.

Additionally, CRBMs can be easily updated and re-trained, by iterating over the new provided information. Here an on-line training approach is applied, where new produced data is used to update the models. Either in the general model scenario or the localized models scenario, data is buffered and fed into the CRBM for training periodically. While localized models just keep their models updated, the Cloud scenario requires transmitting its models to the edge. Figure 5.6 shows the updating schema between edge-cloud-edge on the Cloud scenario.

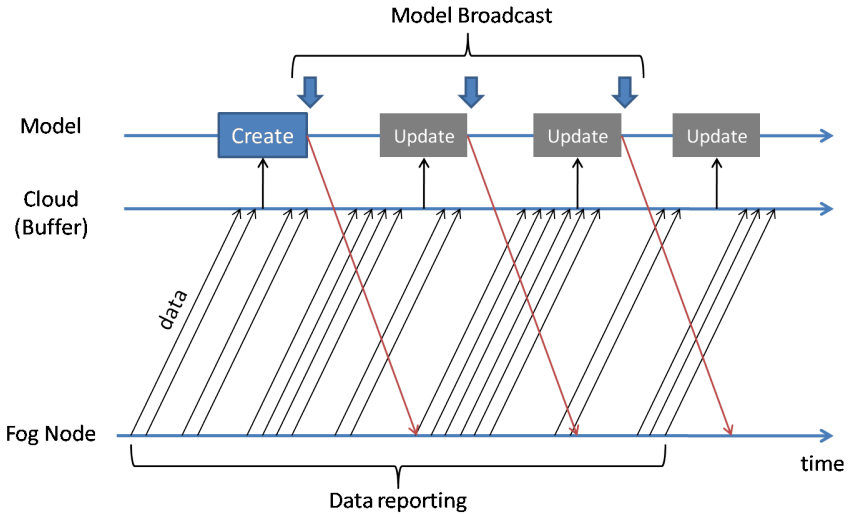


Figure 5.6: Updating models in the Cloud-training scenario

5.3.3 *Data Distribution Algorithms*

One of the principal contributions of this architecture, is the inclusion of a real-time data distribution algorithm with enhanced resilience against back-haul connectivity issues, to avoid data to be lost under connectivity outage periods, and to favor distributed data modeling in the Edge and intermediate Fog nodes.

The algorithm is based on the Fog computing paradigm and node hierarchy between the Edge and the Cloud, to allow data collection and modeling on Fog nodes when suffering connectivity issues, then push data and models towards the Cloud for further analysis and storage when connectivity allows it.

Data distribution among layers is driven by two key algorithms presented as Algorithm 1 and 2, in charge of the operations for inserting data and pushing data, present on each Fog node. The System State monitor, placed in the Cloud level, is responsible to keep the information of all Fog nodes, including network addresses and connection state, also the overall timestamp of the system. The Cloud and intermediate Fog nodes maintain timestamp tracking in order to flag those batches of data arriving late, this is, belonging to a push request previous to the one in course.

Each Fog node contains its own buffer where data is collected. Nodes collecting data from different nature can possess different buffers, to be processed each one independently from the other ones. For simplicity, here nodes with one buffer of data are being shown, but this process can be replicated for as many data streams as necessarily. Then, each buffer is processed when reaching its capacity.

The *Insert Data Algorithms* (Alg. 1) is in charge of processing the data incoming into the Fog node. For each stream of data arriving to a Fog node, the algorithm if there is an already existing buffer dedicated to it, otherwise it creates one, and stores data until the buffer is full. Once the buffer capacity is reached, the content is sent to the Data Synchronization Module, where it is prepared to be pushed towards the upper levels. At this point, the overall system timestamp is updated, and all the sibling Fog nodes are requested to process their buffers into their Data Synchronization Modules. At this step, if a node is unreachable, its state changes to "disconnected". After all the registered buffers are processed, the algorithm notifies the Control Plane on the Cloud level or upper Fog node (in the algorithm referred as "Dispenser") that a processing window has been performed.

The *Push Data Algorithm* (Alg. 2) is in charge of pushing the data when a buffer reaches its capacity, or a Fog node is requested to do that as a result of a sibling node's buffer reaching its capacity. This algorithm is called from the *Insert Data Algorithm* when the aforementioned conditions are met. On the Cloud level (or intermediate Fog nodes), data received from each child node marked as "connected"

is processed and flagged as "ordered", while for nodes marked as "disconnected" at some point, the arrival timestamp of the data is checked by comparing its arrival time versus the overall timestamp, then marked as "disordered" if corresponding. For batches of data marked as "disordered", it is responsibility of the analytics engines to decide whether refuse or process them. For the current scenario, CRBMs can decide whether to reorder data to have available as much data as possible, or refuse it avoiding to increase the load on the Cloud or intermediate nodes. Here it is decided to maintain the last batches of ordered data as input for the CRBMs.

Algorithm 1 Fog node receive data

```
1:  function insertData(stream, body)
2:      Input: Data Stream name stream, Json data body
3:      Output: Http Response Status
4:      data_array ← getStreamDataArray(stream)
5:      if data_array = null then
6:          data_array ← addStreamDataArray(stream)
7:          setStreamState(stream, Config.ip)
8:      end if
9:      data_array.putDocument(body, timestamp)
10:     if data_array.full() then
11:         pushBuffer(data_array, state, stream)
12:         updateTmstp(stream, timestamp)
13:         for all node in state.nodes do
14:             node.pushBuffer(node.stream.data_array,
    node.stream.state, stream)
15:             if error then
16:                 setStreamState(stream, node, false)
17:             end if
18:         end for
19:         Dispenser.postCollect(stream, tmstp)
20:     end if
21:     return Http.Status(201)
22: end function
```

5.4 SYSTEM COMPONENTS

In this section it is described in detail the components that are part of the system architecture described in Section 5.3.

5.4.1 *Fog Node Control Plane*

This component runs in each Fog node, in a completely decentralized mode. It manages the system state through the use of the distributed

Algorithm 2 Fog node push buffer data function

```
1: function pushBuffer(data_array, state, stream)
2:     Input: Node data array data_array, node state state, Data
   Stream name stream
3:     if state.connected then
4:         storeArrayOfData(data_array, stream)
5:     else
6:         for all data in data_array do
7:             created_at ← data.get(created_at)
8:             if created_at > state.tmstp then
9:                 storeData(data, stream, true)
10:            else
11:                storeData(data, stream, false)
12:            end if
13:        end for
14:    end if
15: end function
```

system state component described later in this section. It is implemented as a web component that can be interfaced using REST APIs, and it implements the core algorithms described in Section 5.3.3. The REST APIs are composed of a Servlets Container and a REST Engine. As a HTTP Web Server and Java Servlet container it is used Jetty [18], which is a pure Java-based HTTP server and Java Servlet container. Jetty is often used for machine-to-machine communications, usually within larger software frameworks. As a REST Engine (JSON processor) it is used Jackson [17], which is a high-performance suite of data-processing tools for Java, including the flagship JSON parsing and generation library, as well as additional modules. The Jackson Project also has handlers to add data format support for JAX-RS implementations like Jersey.

5.4.2 *Data Store and Data Synchronization Module*

A distributed data store is used to keep track of all the Fog nodes produced data. For that purpose, CouchBase [4] has been chosen because it provides the benefits of NoSQL data stores (highly distributed, high-availability properties, scalable), and it is document oriented (which fits well for many different data sources and formats). Queries on the data are available using a query DSL. The mechanism to send queries to the platform has been integrated in the API that resides in the central Cloud location.

The data synchronization between Fog nodes and Couchbase is done by the Data Synchronization Module that is deployed with CouchBase Mobile [6]. Couchbase Mobile is composed of Couchbase

Lite [5], an embedded database that manages and stores data locally on the Fog nodes, and the Sync Gateway [7] that provides synchronization between Couchbase Lite and Couchbase Server.

Couchbase has native support for JSON documents. Each JSON document can have a different structure, and multiple documents with different structures can be stored in the same CouchBase bucket. Document structure can be changed at any time, without changing other documents in the database. A Bucket is defined as the owner of a subset of the key space of a Couchbase cluster. These Buckets are used to distribute information effectively across a cluster. A Bucket is equivalent to a database. A common practice is to store documents of different nature on different buckets. The architecture has the ability to store different data nature in different buckets via the Control Plane in the Fog nodes. For the work presented in this paper one bucket has been defined since all the data have the same nature.

5.4.3 *Managing the Distributed System State*

System State is the module that provides the information on the status of all the components of the architecture at all times and is implemented using etcd [12]. Etcd is a distributed reliable key-value store that is automatically replicated with automated master election and consensus establishment using the Raft algorithm, all changes in stored data are reflected across the entire cluster, while the achieved redundancy prevents failures of single cluster members from causing data loss. Etcd also provides service discovery by allowing deployed applications to announce themselves and the services they offer. Communication with etcd is performed through an exposed REST-based API, which internally uses JSON on top of HTTP.

5.4.3.1 *Analytics Engine: local in Fog nodes or global in Cloud*

Both Fog nodes and Cloud have a module for performing the proposed analytics. Such module is in charge to buffer the data for training and updating CRBM models, store the current CRBM model in each situation, forecast data using the models, also to transmit models from Cloud to Fog nodes using the aforementioned REST API. Both Fog and Cloud nodes have available the same implementation, ready to receive data for training/updating or forecasting, and to produce or receive a trained model. The analytics framework is hooked to the data stream, buffering data for periodic model updates. Once a model in the Cloud is trained or updated, it is transmitted to the corresponding Fog nodes. If the model is trained or updated in the Fog nodes, models are kept locally, as explained previously in section 5.3.2.

The analytics and machine learning framework for training CRBMs is created using R from the Comprehensive R Archive Network [69]. The implementation of the CRBM methods is obtained from our im-

plementation in R[1], based on the G. Taylor's original approach. Communication between the REST API and the analytics is built using the package R-Plumber [78]. While R-Plumber manages the connections between API calls and handler functions, the well-known R packages JSONlite [60] and HTTR [82] are used to serialize CRBM models (a set of matrices) and transmit them.

5.5 EVALUATION

In this section we present 5 different experiments that illustrate the behavior of the data distribution algorithms and the traffic modeling component. Across the different experiments, we show how the Cloud-centric learning strategy can lead to incomplete datasets passed along to the analytics engine. This, in turn, results in biased and inaccurate models due to potential back-haul connectivity issues. Therefore, the experiments show the advantages of a decentralized Fog-based learning strategy to improve the accuracy of the models and protect the system against connectivity outages.

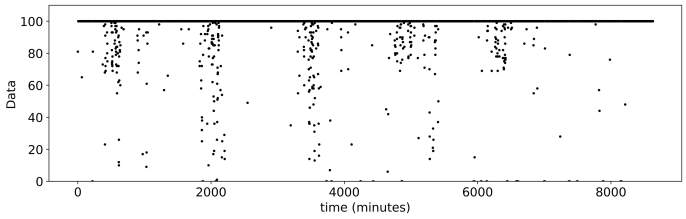
5.5.1 *Methodology*

The data distribution layer was tested under two different conditions: internal memory buffers of 100 elements (small buffer, continuous synchronizations between the Fog node layer and the Cloud components), and 10,000 elements (large buffer, reduced communication patterns between the Edge and Cloud).

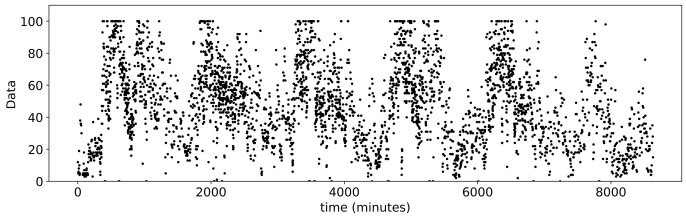
Deployed antennas cover different traffic areas where traffic can be aggregated by coverage zone (using all received data), or by traffic zones (discriminating received data by data source location) if specific streets or delimited traffic areas are to be studied separately. Here we emulated six different antennas (and their corresponding Fog nodes), distributed across the metropolitan area of Barcelona, covering each a different traffic area. Some of the coverage areas are slightly overlapped to guarantee that all the territory got covered by at least one antenna. In all experiments, the FCD data was traversed over time, re-creating the original sequence of events. At each step, data sample was sent to its corresponding nearest antenna. The distribution of data across each Fog node for a configuration corresponding to a node local buffer size of 100 data samples can be seen in Figure 5.7.

Different patterns of network connectivity issues between the Fog nodes and Cloud components were emulated. Network failures were modeled using a mean time between failures that follows a random LogNormal distribution. Different configurations were used, with mean values of 20, 30 and 40 minutes between failures. Figure 5.8 shows the actual time between failure distribution used in the experiments. We

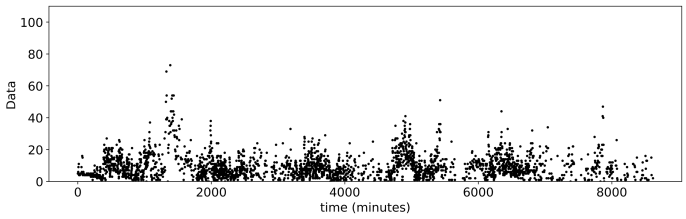
1 https://github.com/josepllberral/machine-learning-tools



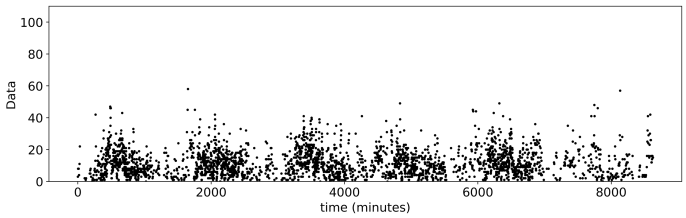
(a) Fog node 1



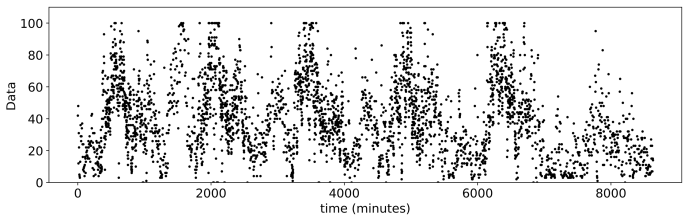
(b) Fog node 2



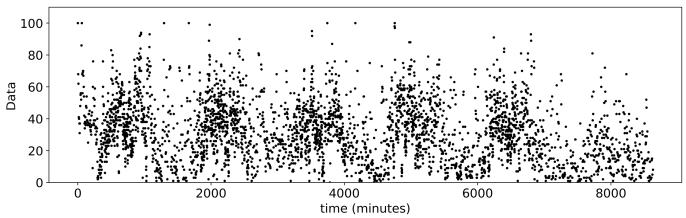
(c) Fog node 3



(d) Fog node 4



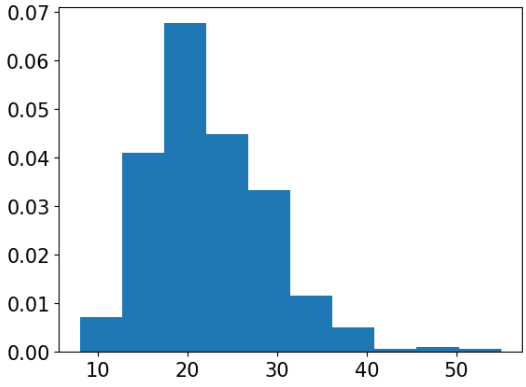
(e) Fog node 5



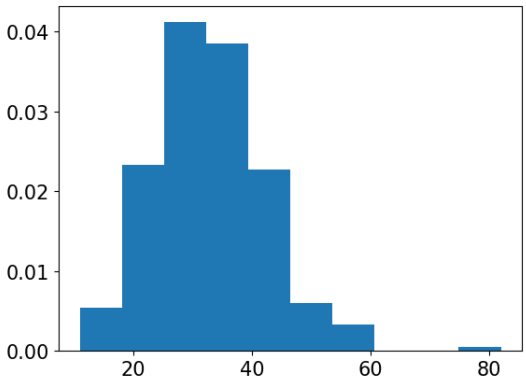
(f) Fog node 6

Figure 5.7: Data processed by each Fog node with 100 buffer items

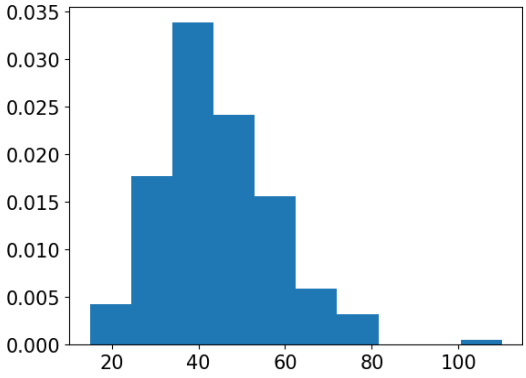
also modeled the duration of each connectivity failure: in this case, we considered another LogNormal distribution with mean value of 10 minutes.



(a) Failure frequency - 20 min



(b) Failure frequency - 30 min



(c) Failure frequency - 40 min

Figure 5.8: Distribution of actual connectivity outages simulated between the Fog nodes and Cloud layers. It follows a random LogNormal distribution

Finally, we also modeled the number of nodes affected in every connectivity outage, In this case, an exponential distribution was used for each node, resulting affected all nodes with values above 0.5 (namely,

low impact scenario). We also considered a second case in which the impact of each failure was significantly higher, representing an scenario in which extreme connectivity issues affect the city. For that scenario, the threshold to decide if a node was affected by the backhaul connectivity outage was set to 0.2 (namely, *high impact* scenario).

5.5.2 *Validation dataset*

With the purpose of validating the current architecture, we have chosen a traffic monitoring and prediction case of use, for a set of 118 vehicles reporting their speed to the Fog node at different moments. We are using real traffic logs from around one week of "Floating Car Data" (FCD) collected in the city of Barcelona, monitoring the traffic in 6 different locations, with 45094 records for 6 days. The collected features include, for each specific Fog node, the number of cars reporting their presence to each node and the average speed for those cars, also each record also includes the position for the Fog node collecting it, and the timestamp. The two most relevant features to be learned and forecasted are traffic volume and average speed, with an average of 9 cars per record and between of 1 to 47 (when no cars are reporting, no record is included in the dataset), and average mean speeds of $19km/h$ and values between $0km/h$ and $124km/h$. Notice that records with speed limits higher than $60km/h$ are in the 99^{th} percentile so we can consider them outliers to prevent them to alter our analytics (also technically the maximum allowed speed in the city is $50km/h$); reducing those records with $speed > 60$ to just 60 keeps the average mean speed to $19km/h$ and values between $0km/h$ and $60km/h$.

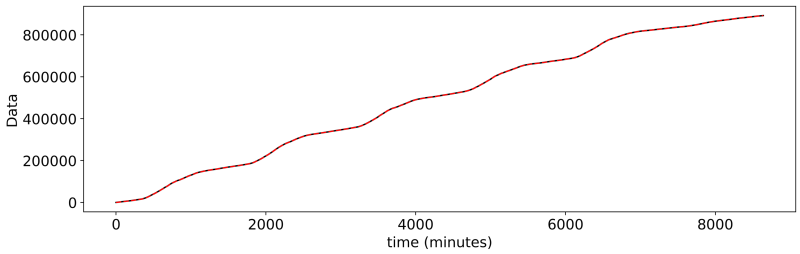
5.5.3 *Evaluation Infrastructure*

The experiments were run in a cluster of 8 servers, each featuring two Xeon E5-2630v4 (broadwell) processors, clocked at 2.20GHz. Each node counts with 128GB of DDR4-2400 R ECC RAM. All nodes were interconnected using a non-blocking 10GbE switching fabric. Although an external NFS folder was mounted on the systems, it was not used as a backend for the experiments. Instead, all data was stored locally using four 7.2K rpm 2TB SATA HDDs per nodes, mounted as four independent volumes.

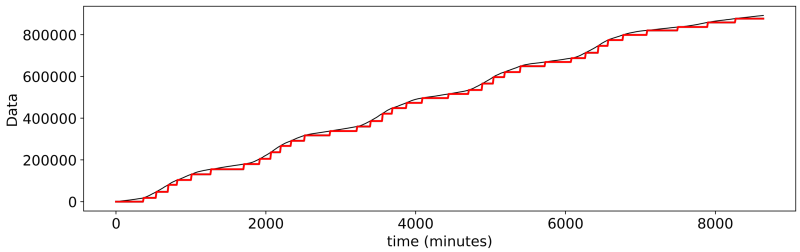
The application was deployed using Docker [56] containers that encapsulated every system component. Docker containers provided all the necessary elements for the evaluation: network isolation, communication and network connectivity control of every component that runs as an isolated process.

5.5.4 *Experiment 1: Percentage of generated data available in Cloud*

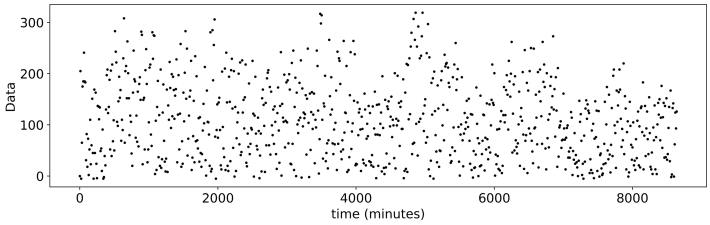
In this first experiment we perform an exploratory analysis of the architecture behavior and algorithms, from the point of view of the ratio of the data available in the cloud (stored in the Data Store) versus the data processed on the Fog nodes.



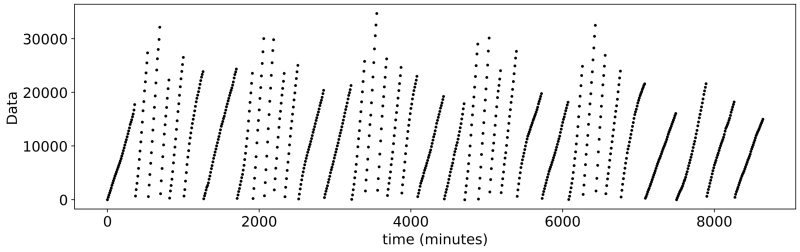
(a) Buffer 100 items - No connetivity issues - accumulate



(b) Buffer 10k items - No connetivity issues - accumulate



(c) Buffer 100 items - No connetivity issues - difference



(d) Buffer 10k items - No connetivity issues - difference

Figure 5.9: Representation of data in Cloud over time without connectivity issues. Increasing buffer sizes in Fog nodes, the communication pattern between the Fog nodes and the Cloud is changed, with less frequent but more intense network traffic bursts when the buffers are larger, and at the same time, delays in propagation are also increased. (Experiment 1)

Figure 5.9a and Figure 5.9b show the cumulative data processed along the time by the Fog nodes and stored in the central Cloud for buffers of 100 and 10k items respectively. As it can be observed, for larger buffers there is a slight sawing in the graph because until the buffer is not filled and launches the processing of all the nodes, which takes more time than for smaller buffers, the processed data will not upload to the Cloud.

From the point of view of the difference between the data generated and the data available in the Cloud, Figure 5.9d shows this effect of buffer filling more clearly showing in a linear way the increase of the difference until it falls through the buffers processing. Figure 5.9c validates that the difference between data generated versus data available in the cloud is small for smaller buffers, for this reason the effect of the buffer processing is blurred.

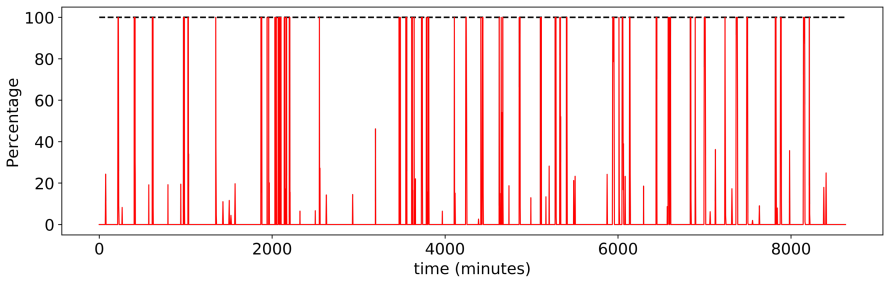
Notice that the Figure 5.9a and Figure 5.9b are not linear, for the reason that there is less data volume at night.

5.5.5 *Experiment 2: Impact of connectivity issues - data affected*

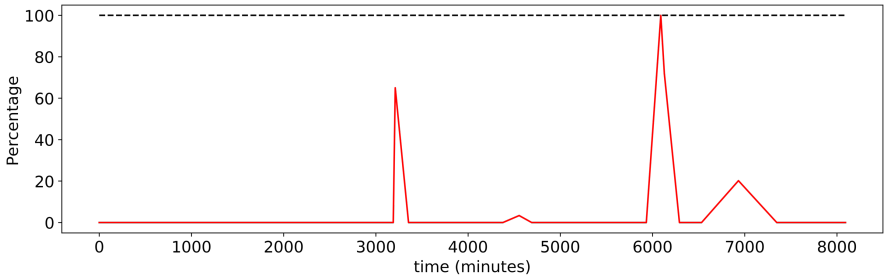
In this second experiments we visualize the amount of data affected by connectivity issues, and therefore delayed more than desired, on the proposed scenarios, showing how in some occasions percentage of losses continuously reaches 100%, always depending on the buffer dimensions. This is the data that at some moment could not be delivered to the Cloud when expected either because the node was disconnected when it went to process its full buffer or because it could not be notified by another node to process it. Realize that may include data delivered in order and data not delivered in order.

Figure 5.10a and Figure 5.10b show the percentage of data affected by connectivity issues with a frequency of next failure of 20 min for Fog nodes with buffers of 100 versus buffers of 10k items. As can be seen, with larger buffers there is less affected data, this is due to the fact that there is less volume of buffer processing and therefore less probability that the Fog nodes are disconnected at that moment. On the contrary, with smaller buffers many times more than the buffer is processed the Fog node is disconnected.

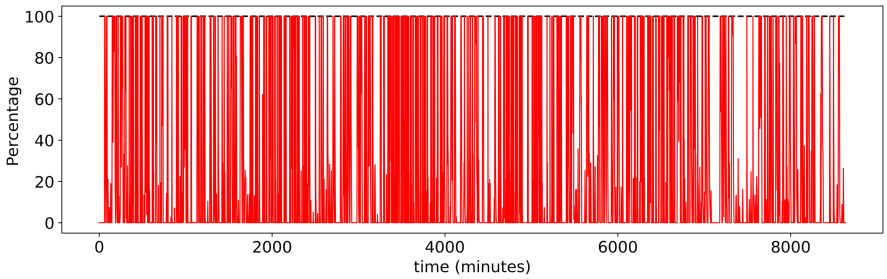
Figure 5.10c and Figure 5.10d show the percentage for the same frequency of next failure (20 min) but incrementing the number of nodes failing. As described in Section 5.5.1, this is done modifying the random exponential function decision value. Incrementing the number of failing nodes to reach a high impact situation can be observed as for small buffers the effect is much greater that for larger buffers. The density of affectation is also greater between nodes with the same buffer size, figure 5.10a and figure 5.10c more affected than for figure 5.10b and figure 5.10d.



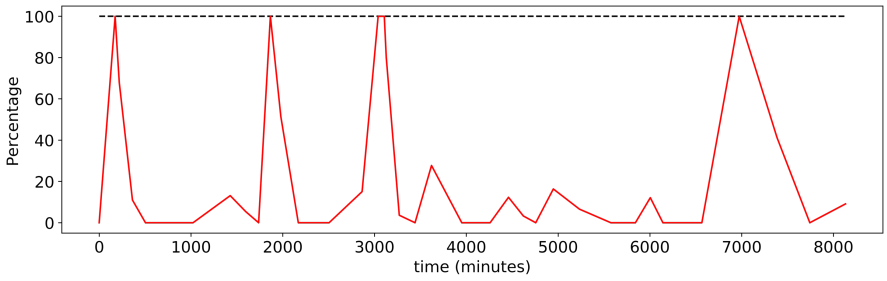
(a) Buffer 100 items - 20 min - low impact



(b) Buffer 10k items - 20 min - low impact



(c) Buffer 100 items - 20 min - high impact



(d) Buffer 10k items - 20 min - high impact

Figure 5.10: Fraction of data sitting in the Fog node Layer is affected by backhaul connectivity issues over time. Simulated time between errors following a random LogNormal distribution with mean values 20min. May include data delivered in order and data not delivered in order to the Cloud. In Low impact scenario, few Fog nodes are affected by the connectivity issues. In the high impact scenario, the scope of connectivity outages is larger. (Experiment 2)

5.5.6 *Experiment 3: Impact of connectivity issues - data delivered out of order*

In this third experiment, we revisit again the data loss ratios in different scenarios, considering only the data delivered out of order from the point of view of the Cloud. The evaluation carried out in this experiment accounts only for the sets of data that could not be delivered in order, this is important for cloud analytics because it can be used to define confidence intervals on existing data in the data warehouse. Notice that the amount of data disordered will be always less than the data affected by connectivity issues (Experiment 2) because is not taken into account the data in the buffers when a Fog node has to process its full buffer and it is disconnected.

Figure 5.11a and figure 5.11b show the percentage of data disordered with a frequency of next failure of 20 min for Fog nodes with buffers of 100 versus buffers of 10k items. As in Experiment 2, with larger buffers there is less disordered data due to the less volume of buffer processing and with smaller buffers more disordered data reach the Cloud.

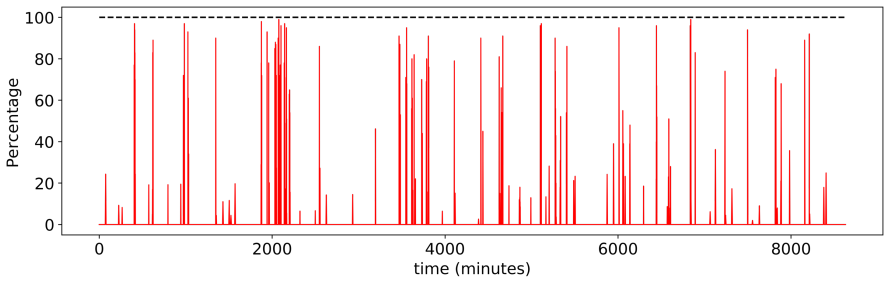
In the same way, under a high impact situation figure 5.11c and figure 5.11d show the same behavior as Experiment 2, for smaller buffers the disorder is much greater that for larger buffers. The density of disorder is also greater between nodes with the same buffer size figure 5.11a and figure 5.11c more affected than for figure 5.11b and figure 5.11d.

As previously explained if overall density between this Experiment and Experiment 2 is compared the density in the figures in Experiment 2 are greater.

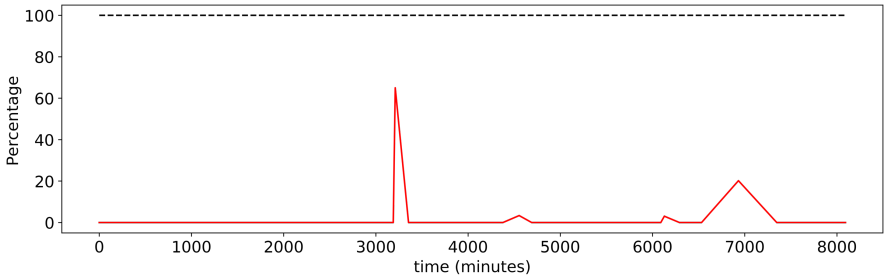
5.5.7 *Experiment 4: Impact of connectivity issues - data contaminated by out of order deliveries*

In this experiment we evaluate all data that has been affected by out of order delivery, directly or indirectly. Here we account the amount of data that has been contaminated (incorrect mix will lead in bad training) either because it was delivered out of order or because other data that should have been mixed with it could not be pushed on time. This is important, not for real time, but to support the idea that building models in the Fog nodes are much more accurate that building models in the Cloud.

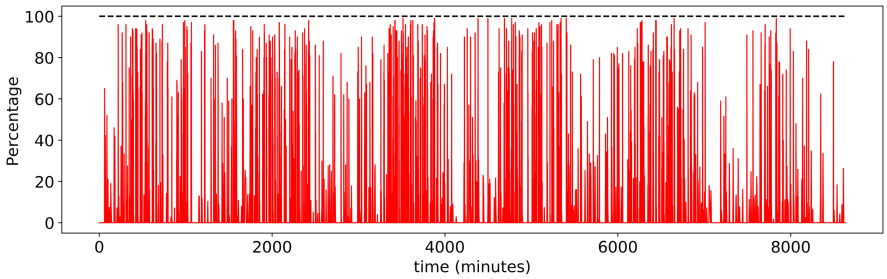
Following the evaluation of the previous experiments (Experiment 2 and Experiment 3), figure 5.12a and figure 5.12b show with a frequency of next failure of 20 min for Fog nodes with buffers of 100 versus buffers of 10k items the percentage of data affected. The data with larger buffers there is less affected than smaller buffers due (as in previous experiments) to the less volume of buffer processing.



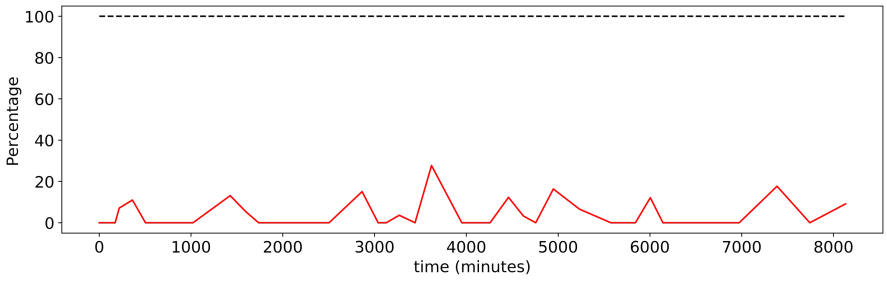
(a) Buffer 100 items - 20 min - low impact



(b) Buffer 10k items - 20 min - low impact



(c) Buffer 100 items - 20 min - high impact



(d) Buffer 10k items - 20 min - high impact

Figure 5.11: Fraction of data sitting in the Fog node Layer that is delivered *out-of-order* to the Cloud layer because back-haul connectivity issues over time. Simulated time between errors following a random LogNormal distribution with mean values 20 min. Only includes data not delivered in order to the Cloud. In Low impact scenario, few Fog nodes are affected by the connectivity issues. In the high impact scenario, the scope of connectivity outages is larger. (Experiment 3)

As previous experiments, under high impact situation figure 5.12c and figure 5.12d show for smaller buffers much more affectation than for larger buffers. Coherently with the rest of experiments . The density of affectation is also greater between nodes with the same buffer size figure 5.12a and figure 5.12c than for figure 5.12b and figure 5.12d.

This experiment presents the highest density of affected data compared with Experiment 2 and Experiment 3. The amount of data contaminated by out of order deliveries, directly or indirectly is the highest.

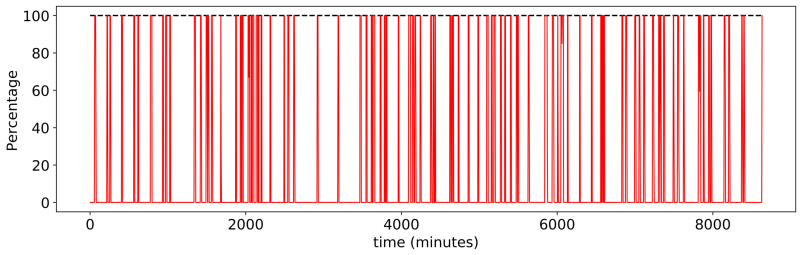
5.5.8 *Experiment 5: Traffic Modeling and Forecasting. Centralized vs Distributed.*

Here we evaluate the modeling and forecasting methodology, to validate how the proposed architecture allows modeling the input data. As previously explained, the principal case of use consists on predicting road traffic properties (i.e. volume of traffic and average speed) per Fog node. We considered two different scenarios: a centralized training process where all data is collected to create one general models, and a distributed training process where each Fog node produces its local model.

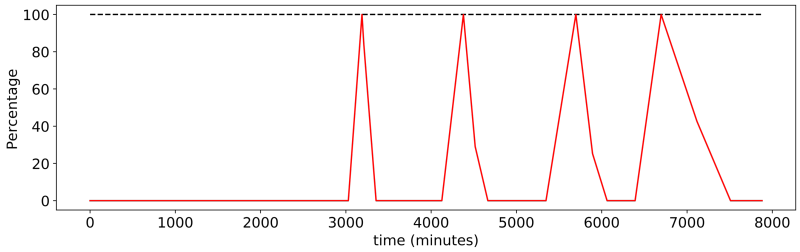
In the following experiments we consider the scenario where data is collected on the Cloud and a CRBM model is created, and the scenario where the CRBM is trained in the Fog nodes. While the first scenario considers no connection failures, becoming the best scenario, the second considers that local models are independent to the ratio of connection failures. In case of no failures, the both scenarios could be performed on the Cloud: a global model and individual models for each node from received data. In case of failures, we can rely on locally trained models, not depending on failures. Then here we show how a global model behaves versus local models.

The CRBM has been tuned after repeated experiments looking for the best hyper-parameters, here hidden units = 30, learning rate = 0.01, momentum = 0.8 and number of training epochs = 4000, also data is aggregated in vehicles and average speed per hour. The historical window kept for prediction is 3 hours, and the forecasting is produced through 30 iterations of Gibbs sampling.

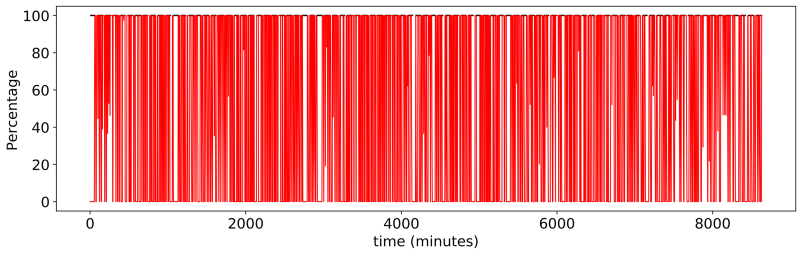
Warm up time for initial CRBM training, also CRBM update (retraining) periodicity is set up to 24 hours, after experimenting with different periodicities, being 24 hours the best update interval. Note that real data displays a daily (24 hour) repetitive pattern, so training the model each 24 hours ensures a fairly balanced set of observations. Additionally, the speed limit has been limited to $60km/h$ (99 percent of observations) to neutralize outliers (for all $s > 60$, it is set to 60), also noticing that maximum speed allowed inside the city is $50km/h$,



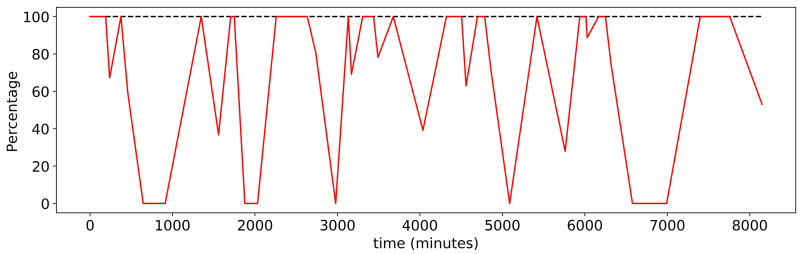
(a) Buffer 100 items - 20 min - low impact



(b) Buffer 10k items - 20 min - low impact



(c) Buffer 100 items - 20 min - high impact



(d) Buffer 10k items - 20 min - high impact

Figure 5.12: Fraction of data sitting in the Fog node Layer that is delivered in *incomplete state* to the Cloud layer because backhaul connectivity issues over time. Data delivered in incomplete state is data that although it is delivered in-order to the Cloud, it contains missing parts that could not be delivered in time. This aspect is important because learning models can get biased because of lack of completeness in the data seen. Simulated time between errors following a random LogNormal distribution with mean values 20min. Only includes data delivered in order to the Cloud, but with missing parts. In Low impact scenario, few Fog nodes are affected by the connectivity issues. In the high impact scenario, the scope of connectivity outages is larger. (Experiment 4)

and those values could be considered towards anomaly detection for traffic law enforcement in future works.

Table 5.1 shows (on the two top tables) the average Relative Absolute Error (RAE, a.k.a. Mean Absolute Percent Error) for each node using a global model, trained and updated using all nodes information, against using local models for each node, trained and updated only with local data. We distinguish the complete error and the 95^{th} percentile error, where we focus on the error for the 95 percent of cases, as we observed that the error distribution produces a long tail, altering the perception of the expected error. As CRBM has a stochastic component, experiments have been run 10 times, and computed the average of results. Figure 5.13 also displays the averaged values in right column for better comprehension.

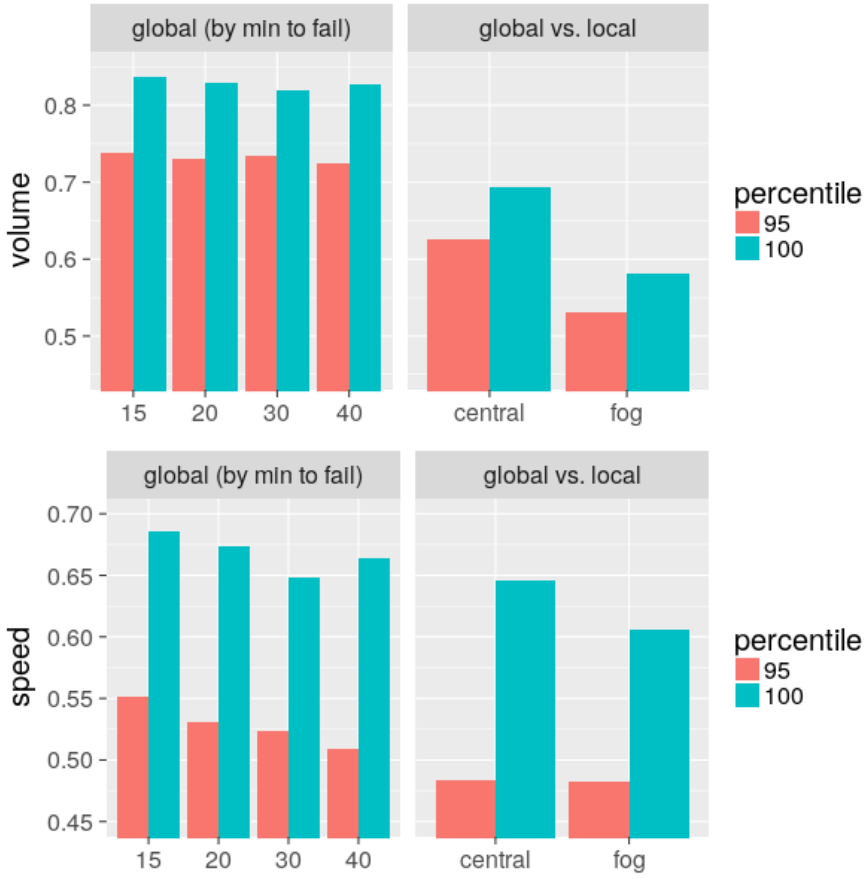


Figure 5.13: Average RAE for values on Table 5.1

We observe that local models produce better predictions than a general model, indicating that each node has enough data for itself to train a decent predictor. Such effect would allow a single node to train or update its CRBM model in case of disconnection from the cloud. On the other side, management systems on the cloud could use the CRBM model to estimate the telemetry from the disconnected

Number of received inputs from each Fog node						
node1	node2	node3	node4	node5	node6	total
8614	8613	5614	5712	8575	7966	45094

	Centralized modeling (One general model)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.1830406	0.5640958	1.0406101	0.9656460	0.6329655	0.7736946	**0.6933421**
average speed	0.5876019	0.4058793	1.7793430	0.4139850	0.3334604	0.3530163	**0.6455477**
traffic volume (95^{th})	0.1706984	0.5323832	0.8967587	0.8716846	0.5885903	0.6971613	**0.6262127**
average speed (95^{th})	0.5089208	0.3332018	1.1704180	0.3400571	0.2810072	0.2700172	**0.4839370**

	Fog node modeling (One model per node)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.1555855	0.5851258	0.6656879	0.6936570	0.6192713	0.7624275	**0.5802925**
average speed	0.5436164	0.2711187	1.8857050	0.3521620	0.2454030	0.3343151	**0.6053867**
traffic volume (95^{th})	0.1394112	0.5423981	0.6077561	0.6287912	0.5654928	0.7018941	**0.5309572**
average speed (95^{th})	0.4206425	0.2248818	1.4680658	0.3243317	0.2126657	0.2462449	**0.4828054**

	Centralized modeling (Avg Time to Failure 40 min)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.2886579	0.5971364	1.326132	1.1623673	0.7139560	0.8785106	**0.8277935**
average speed	0.6542469	0.4493091	1.675766	0.4447477	0.3855491	0.3717311	**0.6635583**
traffic volume (95^{th})	0.2618760	0.5700833	1.084638	0.9722413	0.6642756	0.7956382	**0.7247920**
average speed (95^{th})	0.5307863	0.3588561	1.193760	0.3657113	0.3242667	0.2837089	**0.5095150**

	Centralized modeling (Avg Time to Failure 30 min)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.3047327	0.6216623	1.322312	1.1202736	0.6978688	0.8440265	**0.8184793**
average speed	0.6367097	0.4329232	1.609276	0.4533244	0.3696029	0.3841081	**0.6476574**
traffic volume (95^{th})	0.2775941	0.5908583	1.105833	0.9705806	0.6557692	0.8018136	**0.7337415**
average speed (95^{th})	0.5204882	0.3461248	1.233979	0.3884334	0.3201640	0.3288929	**0.5230137**

	Centralized modeling (Avg Time to Failure 20 min)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.3720164	0.6402134	1.354118	1.0958144	0.7135374	0.8028318	**0.8297553**
average speed	0.6804612	0.4302883	1.746649	0.4494588	0.3640312	0.3713571	**0.6737076**
traffic volume (95^{th})	0.3465650	0.6013991	1.092760	0.9151707	0.6730144	0.7521853	**0.7301824**
average speed (95^{th})	0.6023073	0.3551524	1.210088	0.3834238	0.3073565	0.3230715	**0.5302333**

	Centralized modeling (Avg Time to Failure 15 min)						
	node1	node2	node3	node4	node5	node6	average
traffic volume	0.3622710	0.6315523	1.305127	1.1988274	0.7254418	0.7950532	**0.8363788**
average speed	0.6991176	0.4623524	1.757799	0.4513180	0.3718768	0.3693187	**0.6852970**
traffic volume (95^{th})	0.3384074	0.5947155	1.076719	0.9626721	0.6999604	0.7611931	**0.7389445**
average speed (95^{th})	0.5895197	0.3799004	1.317271	0.3986128	0.3206143	0.3012697	**0.5511980**

Table 5.1: Average RAE (for all data and 95th percentile) on collected-data modeling vs local modeling, per node (top tables). Average RAE for forecasting on failure scenarios (bottom tables)

Fog node. Notice that the error varies per node, as each node registers information from a different part of the city: nodes 3 and 4 are located in the periphery, receiving less data than the others and benefiting from the general Cloud model.

Finally, to conclude the modeling evaluation, we consider how communication failures affect the centralized modeling in the Cloud, by considering that the same aforementioned failure conditions apply, "destroying" part of the training/updating datasets. Here we consider that the analytics module is not bound to delayed data synchronization: CRBMs are trained/updated with real-time available data, and data dispatched late (from previous iterations) is discarded, as considering it would imply to keep extra amount of historic data because of the training pre-processing sorting and completing data. Figure 5.1 shows also (in the four bottom tables) the relative absolute error for traffic volume and average speed prediction (for all error and error 95th percentile), for scenarios where the average time to failure is $\langle 15, 20, 30, 40 \rangle$ versus a no-failure scenario, with an average of 10 minutes of failure and exponential chances for a node to be affected. As failures are stochastic, experiments have been run 10 times, and computed the average of results.

As we can see, the error degrades with higher chances of failure. We must take into account that given the several rounds of updates, CRBMs are able to recover from unseen data up to a certain point, depending on the variability of data in each node and the frequency of failures.

5.5.8.1 *Performance at the Edge*

Devices at the Edge are characterized by low consumption and limited performance, as the Fog paradigm focuses on moving high performance computing to the Cloud while "low cost" operations like aggregation and filtering to the Edge. Machine learning modeling and prediction can require high or low amount of resources depending on the used method and the amount of data to be processed. The used CRBMs have the characteristic of being reasonably easy to train in terms of complexity, as data can be split in mini-batches to be passed by the network then computed the gradient difference, i.e. matrix multiplications and subtractions subject to the size of data and CRBM hidden units, then each piece of data is passed enough times until achieve an acceptable accuracy level. As mentioned before, the CRBMs for the current traffic data require 30 hidden units and less than 4000 iteration to achieve lowest error values.

In order to test the viability of the proposed method into a low performance environment for machine learning and aggregations, we have deployed the framework into Edge devices composed by Raspberry Pi 3B (Raspi) micro-computers, with computing power of 4xARM Cortex-A53 @1.2GHz and 1GB RAM memory, with peak power con-

sumption 3.7W (730mA). Each Raspi is prepared as a Fog Node receiving data from a given coverage area.

As shows Table 5.2, aggregating collected data in rounds of 1 hour and re-training local CRBM models in round of 6 hours, require less than 25% of a CPU and $\sim 21 KBytes$ of RAM memory to aggregating and training data, and times are below 5 seconds to perform the process of aggregation and training, also predicting the new batches of data using the local model. Take into account that the impact of the amount of input data affects uniquely the Aggregation process and Prediction process, as the CRBM models receive data normalized in fixed time-steps.

	Data Processing Time (seconds)				Resources	
	Aggregating	Prediction	Training	# of Inputs	CPU (%)	MEM (KB)
Fog node 1	0.07392797	0.09089618	3.776131	1434.8	15.75	20.8906
Fog node 2	0.07263155	0.08739262	3.765014	1434.6	16.22	20.5495
Fog node 3	0.06041346	0.08531480	3.745345	867.2	16.67	20.7227
Fog node 4	0.06405849	0.08406520	3.005376	962.6	17.14	21.5156
Fog node 5	0.07629604	0.08889236	3.762398	1427.2	17.55	21.6471
Fog node 6	0.07259803	0.08957825	3.762089	1313.6	18.00	21.8542

Table 5.2: Average computing resources and time spent per Edge process, given hourly aggregation and prediction rounds, and 6 hour re-training rounds, in each Fog node

5.5.8.2 *Discussion on Modeling*

Complex policies can be developed in the future like multi-level training if needed, where models are created on the Cloud and also locally or intermediate nodes, and in case of failures Cloud models are selectively discarded when transmitted to Fog nodes, local models are pushed to the Cloud to update the centralized ones, or intermediate models are distributed up and downwards. Here we shown that models can be trained in the Cloud from all data, also in lower levels from local data.

In scenarios where partitions can change behavior along time and models need to generalize, being trained with low probability situations to learn about patterns not seen by it but by others, helps to create a non-over-fitted model with less precision but more accuracy. However, in practice, this is something hard to achieve. As seen in the experiments, local models tend to over-fit to their local samples, usually performing better than global ones. So in case Fog nodes are capable to perform the modeling as part of their analytics, another policy would be to train at the edge then push local models to the Cloud for centralized management purposes, and only re-train models on the Cloud for those nodes benefiting from extra data.

It is also in our road-map to plan scenarios where Fog nodes may change location or move (e.g. Fog nodes on vehicles). So contemplat-

ing a policy where local models can be updated with past data from other nodes in that location could improve them.

5.5.9 *Summary of results*

The following is a summary of the most important aspects of the results presented in this Section:

- Experiment 1: Shows the behavior of the data distribution algorithm in terms of amount of data collected by the Fog nodes and not available at the Cloud level. This factor is conditioned by the size of the buffers in the Fog nodes, that in turn translate into delays in propagating data to the Cloud. At the same time, increasing buffer sizes, the communication pattern between the Fog nodes and the Cloud is changed, with less frequent but more intense network traffic bursts when the buffers are larger.
- Experiment 2: Shows the impact of network connectivity issues between the Fog nodes and Cloud layers, resulting in limited data propagation capabilities during the connectivity outage periods. This aspect is evaluated in terms of percentage of data sitting in the Fog node layer that is affected by the connectivity issues. Data affected is, at least, delayed on its delivery to the Cloud layer. The importance of this aspect for the Traffic modeling components of the application is that it may delay the model training process in the case of using a Cloud-centric modeling strategy, what is expected to have limited impact in the overall system. The Fog-centric model is not affected.
- Experiment 3: Similar to the previous experiment, in this case we explore what is the fraction of data that could not be delivered in order to the Cloud layer. A set of data is not delivered in order when other sets of data, both produced by the devices in the edge during the same time range, are not delivered to the cloud in the right order (being the out-of-order set delivered after the other sets). The importance of this aspect for the Traffic modeling components of the application is that it may degrade the quality of the trained model in the case of using a Cloud-centric modeling strategy, what is expected to have moderate impact in the overall system. The Fog-centric model is not affected.
- Experiment 4: Similar to the previous experiment, in this case we explore what is the fraction of data that was delivered in incomplete state to the Cloud layer because of the sets that could not be delivered in time. A set of data is in incomplete state when it is exposed to the trained model without all the sets that were generated in a given time range. The importance of this

aspect for the Traffic modeling components of the application is that it may produce a severe bias in the the trained model in the case of using a Cloud-centric modeling strategy, what is expected to have severe impact in the overall system. The Fog-centric model is not affected.

- Experiment 5: Provides insights on the actual model accuracy across different scenarios, both for the Cloud-centric and the Fog-centric learning strategies. Results show that the Fog-centric approach is more accurate and at the same time more robust than the Cloud-centric approach, capable to be performed in low computing-power devices.

5.6 RELATED WORK

Several cities around the world are involved in projects towards smart-city management. Platforms designed for management of smart cities exist in cities like Nice, France, where the Connected Boulevard [46] platform has been developed to optimize all aspects of city management, including parking, traffic, street lighting, waste disposal, and environmental quality. Also in Santander, Spain, the project Smart-Santander [73], focuses on a European facility for research and experimentation of architectures, technologies and applications for smart cities, but without focusing yet on Fog computing. Further, other cities like Songdo (South Korea), Masdar City (Abu Dhabi, UAE), Paredes (Portugal), Manchester (UK), Boston (US), Tianjin (China) and Singapore, announced smart-city related projects [76]. Although approaches differ on each city, resilient and secure analytics between the edge and data centers are a hot topic, revolving around a coherent and affordable way of management [51]. The previous work [86], it is focused on how Fog computing architectures can improve the deployment of distributed commercial solutions on smart cities where cloud models fall short, through a Barcelona Supercomputing Center and Cisco Systems joint initiative towards a Fog computing deployment in the city of Barcelona.

The appearance of Floating Car Data (FCD) as data source is expected to provide support to many practical use cases in the near future, leveraging Intelligent Transportation Systems telemetry. To complement the current lack of sensorization in cars and communications infrastructure, works like [42] propose the use of smartphones and Wi-Fi hotspots, also [35] proposes crowdsourcing architectures to collect data from smart devices on vehicles for these same purposes, or [36] studies vehicle-to-vehicle networks to handle the expected escalation of FCD data volume.

In the field of treating Floating Car Data, are found works like [88] where the framework RTIC-C presents a high level architecture to deploy traffic analytics, using Map-Reduce approaches for distribut-

ing modeling and processing algorithms. The RTIC-C authors defend the use of big data analytics on traffic data due to its increasing volume and complexity, then they focus on distributing received data for processing on anomaly detection and traffic trend prediction. The presented framework focuses on the data transmission architecture towards receiving traffic data streams properly for being ingested by analytics methods, i.e. localized re-trainable CRBM prediction mechanisms, and could complement high level frameworks like those named here on generalist model scenarios.

Further, while most state of the art frameworks focus on processing traffic data on dedicated HPC Cloud infrastructures, it is proposed to study scenarios where analytics are partial or completely processed in the Edge. Works like [57] and [83] present different traffic modeling approaches, ensembles using bagging and Feed-Forward MLP Neural Networks the prior and Spatial-Temporal Weighted k-Nearest Neighbor the later, producing general models from the aggregated datasets and distributing computation on the Cloud. Also, works like [54] present a methodology where a Stack of AutoEncoders is applied for traffic flow prediction at different granularities with good results for t+1 forecasting. As presented in this work, localized models on the Edge can create in-situ specialized predictors adapted to their coverage area, using re-trainable machine learning models.

Many Fog applications (e.g. event monitoring and forecasting, as is the case on the presented work), rely on data stream processing analytics. [85] presents general models and architectures for Fog data streaming, and analyze the common properties of the most common applications. An overview about device-to-device communication on the Fog can also be found in [38], focusing on the physical plane of such devices. [84] shows how connectivity issues are important in this field, specifically in wireless communications, applying an algorithm to prioritize which of the available data in a given field with interconnected sensors is send to a mobile carrier and how to route it when connections between data provider and the mobile carrier are intermittent and short in time.

The use of machine learning for time series on Fog computing infrastructures and smart cities is relatively new, although data mining on the Internet of Things has been previously planned and discussed, e.g. [39] proposed the different layers of data management on IoT analytics: data collection, data management, event processing and data mining. Also applications of ML management on cities already exists that could easily benefit from edge computing: from management of power grids using machine learning in big cities from grid monitored data [71], to reduction of data transmission in health-care monitoring wearables by performing pattern mining on the edge [47], to illustrate some examples.

5.7 SUMMARY

Among the new technologies emerging around the Internet of Things (IoT) to provide a new whole scenario for Smart City services, Fog nodes become potential hosts for lightweight services on near real-time close to the edge. Thanks to modern high bandwidth networks, those previously centralized services can become decentralized, adapting to the constant changes on modern cities without compromising fidelity towards centralized management and data warehousing systems.

Here it is presented a decentralized application towards smart-cities traffic monitoring and forecasting. The architecture combines a data distribution layer, connecting the Fog nodes with a centralized Cloud focusing on resilience and near real-time communication, and an on-line Machine Learning modeling technique to learn and predict traffic telemetry. The application is designed to be deployed in a Fog-based infrastructure, in which network antennas (e.g. 5G stations) are combined with Fog nodes to provide near real-time computing capabilities.

The presented approach has been validated and tested through five experiments, illustrating the behavior of the data distribution algorithm plus the traffic modeling analytics methodology. And more important, here evidence of the need for a decentralized Fog-based learning strategy are being provided, to improve the accuracy of the trained models and to protect the system against back-haul connectivity issues. It is observed that, when a centralized cloud-based approach is followed, network connectivity outages limiting the Fog-Cloud communications can produce severe impact in the accuracy of Cloud-learned models, as partial data delivery to the Cloud layer misleads the training process, resulting in less accurate models.

The work described in this chapter has resulted in the following main publication:

[66] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll. Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. *Future Generation Computer Systems*, 87:198 – 212, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.05.013. URL http://www.sciencedirect.com/science/article/pii/S0167739X1732678X

6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

In this thesis we presented three complementary steps toward the development of an unified and distributed orchestration layer for data ingestion and processing based on the Fog Computing paradigm for IoT in a moving data sources environment. In contribution one we proposed the characterization of a platform for hosting IoT workloads in the Cloud providing multi-tenant data stream processing capabilities, the interfaces over an advanced data-centric technology, including the building of a state-of-the-art infrastructure to evaluate the performance and to validate the proposed solution. Second Contribution studies an architectural approach following the Fog paradigm that addresses some of the central challenges found in the first contribution, we studied an extension of the model that addresses some of the central challenges behind the converge of Fog and IoT. And in the last contribution we designed a distributed and scalable platform to perform IoT operations in a moving data environment. After the study about data processing done in Cloud environments in the first contribution and the study of the convenience of the Fog paradigm to solve the IoT close to the Edge made in second contribution, we defined the protocols, the interfaces and the data management to solve the ingestion and processing of data in distributed and orchestrated manner for the Fog Computing paradigm for IoT in a moving data environment.

6.1.1 *Hosting IoT data-centric workloads in the Cloud*

The first contribution of this thesis, presented in Chapter 3, consists of a characterization of a platform for hosting IoT workloads in the Cloud providing mutli-tenant data stream processing capabilities, the interfaces over an advanced data-centric technology. The effectiveness of the platform is demonstrated through a implementation and evaluation of the building of a state-of-the-art infrastructure.

IoT devices and sensors generate streams of data across a diversity of locations and protocols that in the end reach a central platform that is used to store and process these streams. Processing can be done in real time, with transformations and enrichment happening on-the-fly, but it can also happen after data is stored and organized in repositories.

The proposed state-of-the-art platform for hosting Internet of Things (IoT) workloads in the Cloud, ServIoTicy, provides multi-tenant data stream processing capabilities, a REST API, data analytics, advanced queries and multi-protocol support in a combination of advanced data-centric services. The main focus is to provide a rich set of features to store and process data through it REST API, allowing objects, services and humans to access the information produced by the devices connected to the platform. Automatically provisioning resources in the cloud for hosting these platforms as a service requires a detailed understanding of all these components and tiers to allow for auto-scaling and dynamic provisioning capabilities. ServIoTicy aims to provide a technological platform for easily creating services based on the Internet of Things (IoT), thus unleashing the full potential of an Internet of Services (IoS) based on the IoT. ServIoTicy proposes an architecture composed of a Web Tier (where resides the REST API), a Data Store, a Data Indexing and a Stream Processing Topology.

Several technologies for implementing the main components of the platform have been used and have been characterized. The NoSQL database Couchbase for the distributed data store, the search engine ElasticSearch for data indexing and Servlets Container and a REST Engine for the Web Tier. We have provided a set or resource actuations through four main HTTP operations to expose the objects and their operations and for the lightweight-interchange format we used the open-standard JSON.

To evaluate the proposed solution, the platform is deployed in two sets of nodes: one set for running the client emulators and one set for running the servers of the system under test.

A number of experiments are executed to demonstrate the effectiveness of the proposed solution with the regards to the three objectives: A detailed workload and resource characterization of the servIoTicy major components (REST API tier, Distributed Data Store and Indexing Engine) from a point of view of scalability with the available re-

sources; An evaluation of the efficiency of CouchBase as a distributed data store in terms of response time delivered wit the load; and An insight on the performance impact of a proper configuration of the memory settings in ElasticSearch. The characterization has revealed interesting details about the three main components involved in the process of storing and retrieving data: the REST API, the data store and the search and indexing engine. By the time the work presented in this thesis was started there were no tools to achieve it and we had to develop our own evaluation system. Currently we find Benchmarking tools for IoT like the widely used TPCx-IoT [29].

6.1.2 *Distribution of data processing under the Fog paradigm*

The second contribution of this thesis, described in Chapter 4, is a definition and development of the components related with interfaces for life cycle management and data processing in a Fog Computing environment. We have validated the relevance and necessity of Fog for IoT under a city environment. The Barcelona City demo shows that a platform like the one shown can efficiently address the majority set of problems for cities.

The design is focused on common needs across verticals and industries. The target was to create a platform that can serve multiple purposes and be ported and reused in other IoT domains. Also the design was guided by four principles: Openness, virtualization, data-app centric model and uniform management and service consolidation. The design principles followed allow Fog computing to elevate IoT from point solutions to manage services at the Edge.

We validated the relevance and necessity of Fog for some of the IoT demanding. Although Cloud is seen as the go-to solution for many IoT-related challenges, we have come to understand that Fog is crucial. Our model offers a common and distributed data fabric across the city for multiple departments (that is, tenants), and is based on a platform that seamlessly combines Cloud computing and Fog computing. We also have introduced an open and converged architecture based on MANO that offers uniform management of IoT services spanning the continuum from the Cloud to the Edge.

We have addressed the development of the systems required for provisioning, deploying, managing, and maintaining the compute, network, and storage resources needed for running Fog Services. We also have demonstrated the strengths of a consolidated service platform for IoT with a demo with the Barcelona City Hall to cover the uses chases to address a set of specific IoT challenges. We examined the requirements of a growing number of urban services where cloud-centric approaches are insufficient and Fog computing is mandatory. We describe use cases (UCs) that were implemented

and demonstrated in Barcelona, providing supporting evidence for Fog computing.

The proposed architecture platform was demonstrated in a co-innovation project with the Barcelona City Council and several industrial partners. We described use cases (UCs) that were implemented and demonstrated in Barcelona, providing supporting evidence for Fog computing. The word described an architectural approach that addresses some of the technical challenges behind the convergent between Fog, Network Functions virtualization and Mobile Edge Computing, bridging the gap between Cloud and Fog. We also introduced a model-driven and service-oriented architecture aligned with the OpenFog Consortium reference architecture.

During the writing of this thesis, the OpenFog Consortium's OpenFog Reference Architecture for Fog Computing has been adopted as an official standard by the IEEE Standards Association (IEEE-SA). The new standard, known as IEEE 1934 [34], relies on the reference architecture as a universal technical framework that enables the data-intensive requirements of the Internet of Things (IoT), 5G and artificial intelligence (AI) applications.

6.1.3 *Distribution and scalability of IoT operations with moving data sources*

Finally, the third contribution of this thesis, presented in Chapter 5, is a proposed architecture for a city-wide traffic service based on the Fog Computing paradigm. Thanks to modern high bandwidth networks and the architectures associated with them (MANO, MEC), services can become decentralized and computation can be done on the Edge.

After the first two contributions where we explore the trade-offs and challenges in the design of architectures for IoT based in Cloud and Fog paradigms we presented a converged Cloud/Fog architecture that addresses the challenges currently IoT is facing and an application running inside. The application is designed to be deployed in a Fog-based infrastructure, in which network antennas (e.g. 5G stations) are combined with Fog nodes to provide near real-time computing capabilities.

The contribution assumed an scenario in which a number of distributed antennas receive data generated by vehicles across the city. In the Fog nodes data is collected, processed in local and intermediate nodes, and finally forwarded to a central Cloud location for further analysis. We proposed a combination of a data distribution algorithm, resilient to black-haul connectivity issues, and a traffic modeling approach as an example of application running in the architecture.

In order to evaluate the architecture proposal, a number of experiments have been executed in a containerized environment, illustrating the behavior of the data distribution algorithm plus the traffic

modeling analytics methodology. We leveraged real traffic logs from one week of Floating Car Data (FCD) generated in the city of Barcelona by a road-assistance service fleet comprising thousands of vehicles. FCD was processed across several simulated conditions, ranging from scenarios in which no connectivity failures occurred in the Fog nodes, to situations with long and frequent connectivity outage periods.

For each scenario, the resilience and accuracy of both the data distribution algorithm, and the learning methods were analyzed. Results showed that the data distribution process running in the Fog nodes is resilient to back-haul connectivity issues and is able to deliver data to the Cloud location even in presence of severe connectivity problems. Additionally, the proposed traffic modeling and forecasting method exhibited better behavior when run distributed in the Fog instead of centralized in the Cloud, especially when connectivity issues occur that force data to be delivered out of order to the Cloud.

6.2 FUTURE WORK

The work performed in this thesis opens several interesting proposals that could be explored as part of future work.

- The architecture proposed in the third contribution could be expanded in different ways for different needs. Parent-children structures for Fog nodes could be implemented for applications demanding precalculation in different layers. Also tree structures could be implemented for applications where analytics need to perform in a tree structured path. We could also introduce territorial criteria, sensor data received at the Edge nodes may be provided by sensors installed at devices (fixedly or movably) located at particular territorial units, for example, different sets of sensors may be located at different street segments in a city to produce sensor data characterizing said street segment. Sensors located at street segments forming a complete street may provide sensor data to Edge nodes with same middle node as parent node. Middle nodes covering complete streets of same district may be configured as children nodes of same middle node at an above level in the fog computing system. This manner, different models covering (or characterizing) different territorial units: street segment, complete street, district, city, etc. may be analyzed at corresponding Fog levels.

- The architecture presented in the third contribution is aware of the state of the whole system via the System State module. This System State is distributed in the Cloud of the architecture. It would be one step further to down the Distributed System State to the Edge. Different possibilities open to this option, depend-

ing on the computing capacity of each node, it could have a copy of the system state in which they support it. Another possibility in a multi-layer/tree Edge mapping would be to have the system state of your neighbors nodes. We could also have a set of nodes with the system status and mechanisms of knowledge of the proximity of nodes to consult the closest.

- Another path to research would be to increase the data locality for IoT architectures. The global data store is sitting in the Cloud and would be a new path to research to explore the options to allocate this Data Store to the Edge to open opportunities for location global data near to Edge computation. Addressing this scenario, the Fog nodes would not only have their primary data and analytics, but would also have close the global data of the general and/or partial analytics that now reside in the Cloud. How to reach this data locality for different applications residing in the architecture would be an opportunity to improve computation where it is necessary.

- Many IoT systems are mission critical and pose grave risks if hacked, so cybersecurity and data privacy are one of the most important areas to improve and research. Blockchain, and the combination of cryptographic processes behind it, offers an ideal approach. Because blockchain is built for decentralized control, a security scheme based on it should fit with the demands of the Internet of Things. Blockchain is promising for IoT security for the same reasons it works for cryptocurrency: It provides assurances that data is legitimate, and the process that introduces new data is well-defined. Blockchain technology could provide a simple infrastructure for two devices to directly transfer a piece of property such as money or data between one another with a secured and reliable time-stamped contractual handshake. It seems that a great future work field in security for IoT would be the exploration of Blockchain as an option.

BIBLIOGRAPHY

[1] View on 5g architecture. URL https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf. Accessed in: 10-May-2017.

[2] "NFV". URL http://www.etsi.org/technologies-clusters/technologies/nfv. on-line; accessed in: 10-May-2017.

[3] 50 Billion Things, Coming to a Cloud Near You. URL https://blogs.cisco.com/news/50-billion-things-coming-to-a-cloud-near-you.

[4] Couchbase, . URL http://www.couchbase.com/. Accessed in: 10-June-2017.

[5] Couchbase lite, . URL https://goo.gl/g98TzM. "on-line; accessed in: 15-Nov-2017".

[6] Couchbase mobile, . URL https://www.couchbase.com/products/mobile. "on-line; accessed in: 15-Nov-2017".

[7] Couchbase sync gateway, . URL https://goo.gl/VeVMNy. "on-line; accessed in: 15-Nov-2017".

[8] DeviceHive, . URL http://www.devicehive.com/.

[9] Devicehub, . URL http://devicehub.net.

[10] Elasticsearh. URL http://www.elasticsearch.org/. Accessed in: 10-June-2017.

[11] Things in the Networked Society - connected and intelligent. URL https://www.ericsson.com/thinkingahead/the-networked-society-blog/2015/04/27/internet_of_things-in-the-networked-society_connected-and-intelligent_iot/.

[12] Etcd. URL https://github.com/coreos/etcd. "on-line; accessed in: 15-Nov-2017".

[13] evrythng. URL evrythng.com.

[14] Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. URL https://www.gartner.com/newsroom/id/3165317.

[15] Idc forecasts worldwide spending on the internet of things to reach $772 billion in 2018. URL https://www.idc.com/getdoc.jsp?containerId=prUS43295217. Accessed in: 10-May-2017.

[16] IoT Toolkit. URL iot-toolkit.com.

[17] Jackson. URL http://jackson.codehaus.org/. Accessed in: 10-June-2017.

[18] Jetty. URL http://www.eclipse.org/jetty/. Accessed in: 10-June-2017.

[19] Kestrel. URL https://github.com/twitter/kestrel.

[20] Mqtt. URL http://mqtt.org/. Accessed in: 11-October-2017.

[21] Nimbits. URL http://www.nimbits.com.

[22] Openfog consortium, . URL http://www.openfogconsortium.org/. Accessed in: 10-May-2017.

[23] OpenRemote, . URL http://www.openremote.com.

[24] servIoTicy. URL http://www.servioticy.com.

[25] SiteWhere. URL http://www.sitewhere.org.

[26] Stomp, . URL http://stomp.github.io.

[27] Storm, . URL http://storm-project.net.

[28] ThingSpeak. URL https://thingspeak.com/.

[29] Tpcx-iot. URL http://www.tpc.org/tpcx-iot/default.asp. Accessed in: 5-June-2018.

[30] The webSocket API. URL http://dev.w3.org/html5/websockets.

[31] Xively. URL xively.com.

[32] That "internet of things" thing. *RFID Journal*, 2009. URL http://www.rfidjournal.com/articles/view?4986.

[33] "Network Functions Virtualisation (NFV), Management and Orchestration", 2014. URL http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf. on-line; accessed in: 10-May-2017.

[34] IEEE std 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, 2018. URL http://standards.ieee.org/findstds/standard/1934-2018.html.

[35] K. Ali, D. Al-Yaseen, A. Ejaz, T. Javed, and H. S. Hassanein. Crowdits: Crowdsourcing in intelligent transportation systems. In *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3307–3311, April 2012. doi: 10.1109/WCNC.2012.6214379.

[36] S. Ancona, R. Stanica, and M. Fiore. Performance boundaries of massive floating car data offloading. In *2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 89–96, April 2014. doi: 10.1109/WONS.2014.6814727.

[37] Aleksandar Antonic, Kristijan Rozankovic, Martina Marjanovic, Kresimir Pripuzic, and Ivana Podnar Zarko. A mobile crowdsensing ecosystem enabled by a cloud-based publish/subscribe middleware. In *The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014)*, 2014.

[38] O. Bello and S. Zeadally. Intelligent device-to-device communication in the internet of things. *IEEE Systems Journal*, 10(3): 1172–1182, Sept 2016. ISSN 1932-8184. doi: 10.1109/JSYST.2014.2298837.

[39] Shen Bin, Liu Yuan, and Wang Xiaoyi. Research on data mining models for the internet of things. In *2010 International Conference on Image Analysis and Signal Processing*, pages 127–132, April 2010. doi: 10.1109/IASP.2010.5476146.

[40] M. Bjorklund. Yang - a data modeling language for the network configuration protocol (netconf). RFC 6020, RFC Editor, October 2010. URL http://www.rfc-editor.org/rfc/rfc6020.txt. http://www.rfc-editor.org/rfc/rfc6020.txt.

[41] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapè. On the integration of cloud computing and internet of things. In *The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014)*, 2014.

[42] Orazio Briante, Claudia Campolo, Antonio Iera, Antonella Molinaro, Stefano Yuri Paratore, and Giuseppe Ruggeri. Supporting augmented floating car data through smartphone-based crowdsensing. *Vehicular Communications*, 1(4):181 – 196, 2014. ISSN 2214-2096. doi: https://doi.org/10.1016/j.vehcom.2014.08.002.

[43] Xianggao Cai and Xiaola Lin. Forecasting high dimensional volatility using conditional restricted boltzmann machine on gpu. In *IPDPS Workshops*, pages 1979–1986. IEEE Computer Society, 2012. ISBN 978-1-4673-0974-5.

[44] Ivano Cerrato, Alex Palesandro, Fulvio Risso, Marc Suñé, Vinicio Vercellone, and Hagen Woesner. Toward dynamic virtualized

network services in telecom operator networks. *Computer Networks*, 92:380 – 395, 2015. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2015.09.028. URL http://www.sciencedirect.com/science/article/pii/S1389128615003485. Software Defined Networks and Virtualization.

[45] I. Chih-Lin and et al. 5G: Rethink mobile communications for 2020+. *Philos Trans A Math Phys Eng Sci*, 374, 2016.

[46] A. Corsaro. Connected boulevard - it's what makes nice, france, a smart city, September 2014. URL https://goo.gl/sFjCUq. "online; accessed in: 15-Nov-2017".

[47] Harishchandra Dubey, Jing Yang, Nick Constant, Amir Mohammad Amiri, Qing Yang, and Kunal Makodiya. Fog data: Enhancing telehealth big data through fog computing. In *Proceedings of the ASE BigData & SocialInformatics 2015*, ASE BD&SI '15, pages 14:1–14:6, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3735-9. doi: 10.1145/2818869.2818889.

[48] M.Chiang et al. *"Smart Data Pricing"*. Wiley, 2014.

[49] Y. C. Hu et al. "Mobile Edge Computing A key technology towards 5G", 2015. URL http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf. on-line; accessed in: 10-May-2017.

[50] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Compututing*, 14(8):1771–1800, August 2002. ISSN 0899-7667. doi: 10.1162/089976602760128018.

[51] Yu Chao Hu, Milan Patel, Dario Sabella, Nurlt Sprecher, and Valerie Young. Mobile edge computing: A key technology towards 5g, white paper, etsi, 2015. URL https://goo.gl/Qx9W4k. "online; accessed in: 15-Nov-2017".

[52] S. Kitanov and T. Janevski. State of the art: Fog computing for 5g networks. In *2016 24th Telecommunications Forum (TELFOR)*, pages 1–4, Nov 2016. doi: 10.1109/TELFOR.2016.7818728.

[53] E. Lear, R. Droms, and D. Romascanu. Manufacturer usage description specification. IETF draft, 2017. URL https://tools.ietf.org/html/draft-ietf-opsawg-mud-05. Accessed in: May 10, 2017.

[54] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei Yue Wang. Traffic Flow Prediction With Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2014. ISSN 15249050. doi: 10.1109/TITS.2014.2345663.

[55] Benny Mandler, Fabio Antonelli, Robert Kleinfeld, Carlos Pedrinaci, David Carrera, Alessio Gugliotta, Daniel Schreckling, Iacopo Carreras, Dave Raggett, Marc Pous, Carmen Vicente Villares, and Vlad Trifa. Compose – a journey from the internet of things to the internet of services. *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, 0:1217–1222, 2013. doi: http://doi.ieeecomputersociety.org/10.1109/WAINA.2013.116.

[56] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014. ISSN 1075-3583. URL http://www.docker.com. "on-line; accessed in: 15-Nov-2017".

[57] Fabio Moretti, Stefano Pizzuti, Stefano Panzieri, and Mauro Annunziato. Urban traffic flow forecasting through statistical and neural network bagging ensemble hybrid modeling. *Neurocomputing*, 167:3–7, 2015. ISSN 18728286. doi: 10.1016/j.neucom.2014.08.100.

[58] David Mosberger and Tai Jin. Httperf—a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26 (3):31–37, December 1998. ISSN 0163-5999. doi: 10.1145/306225.306235. URL http://doi.acm.org/10.1145/306225.306235.

[59] Stefan Nastic, Sanjin Sehic, Duc-Hung Le, Hong-Linh Truong, and Schahram. Provisioning software-defined iot cloud systems. In *The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014)*, 2014.

[60] Jeroen Ooms. The jsonlite package: A practical and consistent mapping between json data and r objects. 2014. URL https://arxiv.org/abs/1403.2805.

[61] Mell P. and Grace T. The nist definition of cloud computing. national institute of standards and technology 53 (6), 50. 2009.

[62] J. P. Patwardhan, A. R. Lebeck, and D. J. Sorin. Communication breakdown: analyzing cpu usage in commercial web workloads. In *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 12–19, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8385-0.

[63] Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky, and John Domingue. iserve: a linked services publishing platform. In *The 7th Extended Semantic Web Ontology Repositories and Editors for the Semantic Web Workshop*, volume 596, June 2010. URL http://oro.open.ac.uk/23093/.

[64] Juan Luis Pérez and David Carrera. Performance characterization of the servioticy API: an iot-as-a-service data management platform. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 62–71. IEEE Computer Society, 2015. ISBN 978-1-4799-8128-1. doi: 10.1109/BigDataService.2015.58. URL https://doi.org/10.1109/BigDataService.2015.58.

[65] Juan Luis Pérez, Álvaro Villalba, David Carrera, Iker Larizgoitia, and Vlad Trifa. The COMPOSE API for the internet of things. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 971–976. ACM, 2014. ISBN 978-1-4503-2745-9. doi: 10.1145/2567948.2579226. URL http://doi.acm.org/10.1145/2567948.2579226.

[66] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll. Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. *Future Generation Computer Systems*, 87:198 – 212, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.05.013. URL http://www.sciencedirect.com/science/article/pii/S0167739X1732678X.

[67] M. Ptiček, V. Čačković, M. Pavelić, M. Kušek, and G. Ježić. Architecture and functionality in m2m standards. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 413–418, May 2015. doi: 10.1109/MIPRO.2015.7160306.

[68] Yongrui Qin, Quan Z. Sheng, Nickolas J. G. Falkner, Schahram Dustdar, Hua Wang, and Athanasios V. Vasilakos. When things matter: A data-centric view of the internet of things. *CoRR*, abs/1407.2704, 2014. URL http://arxiv.org/abs/1407.2704.

[69] R Core Team. R: A language and environment for statistical computing, 2013. URL http://www.R-project.org/.

[70] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks: An overview. *IEEE Communications Surveys And Tutorials*, 19:855–873, 2017.

[71] C. Rudin, D. Waltz, R. N. Anderson, A. Boulanger, A. Salleb-Aouissi, M. Chow, H. Dutta, P. N. Gross, B. Huang, S. Ierome, D. F. Isaac, A. Kressner, R. J. Passonneau, A. Radeva, and L. Wu. Machine learning for the new york city power grid. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):328–345, Feb 2012. ISSN 0162-8828. doi: 10.1109/TPAMI.2011.108.

[72] Ruslan Salakhutdinov and Geoffrey Hinton. Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes. *Advances in Neural Information Processing Systems 20*, 20:1–8, 2008.

[73] L. Sanchez, V. Gutierrez, J. A. Galache, P. Sotres, J. R. Santana, J. Casanueva, and L. Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–6, June 2013.

[74] Graham W. Taylor and Geoffrey E. Hinton. Factored conditional restricted Boltzmann Machines for modeling motion style. *Proceedings of the 26th International Conference on Machine Learning (ICML 09)*, pages 1025–1032, 2009. ISSN 1520510X. doi: 10.1145/1553374.1553505.

[75] Graham W. Taylor, Geoffrey E Hinton, and Sam T. Roweis. Modeling human motion using binary latent variables. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1345–1352. MIT Press, 2007.

[76] Technology Review. Cities get smarter, 2015. URL https://goo.gl/fSd1mu. "on-line; accessed in: 15-Nov-2017".

[77] Karen Tillman. How many internet connections are in the world? right. now., 2013. URL http://blogs.cisco.com/news/cisco-connections-counter. Accessed in: 11-June-2017.

[78] Trestle Technology. Plumber: An api generator for r, 2017. URL https://www.rplumber.io.

[79] Frank van Lingen, Marcelo Yannuzzi, Anuj Jain, Rik Irons-Mclean, Oriol Lluch Parellada, David Carrera, Juan Luis Pérez, Alberto Gutierrez, Diego Montero, Josep Marti, Ricard Maso, and Juan Pedro Rodriguez. The unavoidable convergence of nfv, 5g, and fog: A model-driven approach to bridge cloud and edge. *IEEE Communications Magazine*, 55(8):28–35, 2017.

[80] Álvaro Villalba, Juan Luis Pérez, David Carrera, Carlos Pedrinaci, and Luca Panziera. serviotic y and iserve: A scalable platform for mining the iot. In Elhadi M. Shakshuki, editor, *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), London, UK, June 2-5, 2015*, volume 52 of *Procedia Computer Science*, pages 1022–1027. Elsevier, 2015. doi: 10.1016/j.procs.2015.05.097. URL https://doi.org/10.1016/j.procs.2015.05.097.

[81] David Carrera Vlad Trifa, Dominique Guinard. Web thing model w3c submision, 2015. URL https://www.w3.org/Submission/wot-model. Accessed in: 11-June-2015.

[82] Hadley Wickham. httr: Tools for working with urls and http. 2017. URL https://cran.r-project.org/package=httr.

[83] Dawen Xia, Binfeng Wang, Huaqing Li, Yantao Li, and Zili Zhang. A distributed spatial-temporal weighted model on MapReduce for short-term traffic flow forecasting. *Neurocomputing*, 179:246–26, 2016. ISSN 18728286. doi: 10.1016/j.neucom.2015.12.013.

[84] Gang Xu, Edith C-H Ngai, and Jiangchuan Liu. Ubiquitous transmission of multimedia sensor data in internet-of-things. *IEEE Internet of Things Journal*, 2017.

[85] S. Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27, 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600840.

[86] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, and Albert Olive. A new era for cities with fog computing. *IEEE Internet Computing*, 21(2):54–67, 2017. doi: 10.1109/MIC.2017.25. URL https://doi.org/10.1109/MIC.2017.25.

[87] M. Ye and L. Cheng. System-performance modeling for massively multiplayer online role-playing games. *IBM Syst. J.*, 45(1):45–58, 2006. ISSN 0018-8670. doi: http://dx.doi.org/10.1147/sj.451.0045.

[88] Jianjun Yu, Fuchun Jiang, and Tongyu Zhu. RTIC-C: A big data system for massive traffic information mining. *Proceedings - 2013 International Conference on Cloud Computing and Big Data, CLOUDCOM-ASIA 2013*, pages 395–402, 2013. ISSN 2378-3680. doi: 10.1109/CLOUDCOM-ASIA.2013.91.